2014-09

Attributes and machine learning for fragment identification and malware analysis

Beneduce, Kristen

Monterey, California: Naval Postgraduate School

NAVAL
POSTGRADUATE
SCHOOL

MONTEREY, CALIFORNIA

THESIS

ATTRIBUTES AND MACHINE LEARNING FOR FRAGMENT IDENTIFICATION AND MALWARE ANALYSIS

by

Kristen Beneduce

September 2014

Thesis Advisor:	Joel Young
Second Reader:	Chris Eagle

Approved for public release; distribution is unlimited

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704–0188

1. AGENCY USE ONLY *(Leave Blank)*	**2. REPORT DATE** 09-29-2014	**3. REPORT TYPE AND DATES COVERED** Master's Thesis 06-01-2012 to 09-29-2014

4. TITLE AND SUBTITLE ATTRIBUTES AND MACHINE LEARNING FOR FRAGMENT IDENTIFICATION AND MALWARE ANALYSIS	**5. FUNDING NUMBERS**
6. AUTHOR(S) Kristen Beneduce	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES

The views expressed in this document are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol Number: N/A.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	**12b. DISTRIBUTION CODE**

13. ABSTRACT *(maximum 200 words)*

This study applies machine learning techniques and novel statistical features for two important classification problems in secure computing: malware detection and file fragment type identification. We observe combinations of information-theoretic and Natural Language Processing features extracted from byte level file content. To the extent possible, we replicate recent studies to validate the use of these features and expand on recent work by combining features from malware to detection to fragment identification tasks and vice versa. By avoiding the use of extracted file signatures and strings, this study contributes techniques that may be more resistant to obfuscation attacks, lead to enhanced prediction rates for zero-day malware files, and improved forensics on broken fragments where file metadata information is not available. We evaluate our results against recent works and report the highest performing algorithms and combinations of features for each task.

14. SUBJECT TERMS Machine Learning, Information Theory, File Forensics, Malware Detection, Digital Fingerprinting, Anomaly Detection	**15. NUMBER OF PAGES** 81
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 **Standard Form 298 (Rev. 2–89)**
Prescribed by ANSI Std. 239–18

THIS PAGE INTENTIONALLY LEFT BLANK

ATTRIBUTES AND MACHINE LEARNING FOR FRAGMENT IDENTIFICATION AND MALWARE ANALYSIS

Kristen Beneduce
Civilian, Department of the Navy
B.A. University of Pennsylvania

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2014

Author:	Kristen Beneduce
Approved by:	Joel Young Thesis Advisor
	Chris Eagle Second Reader
	Peter Denning Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This study applies machine learning techniques and novel statistical features for two important classification problems in secure computing: malware detection and file fragment type identification. We observe combinations of information-theoretic and Natural Language Processing features extracted from byte level file content. To the extent possible, we replicate recent studies to validate the use of these features and expand on recent work by combining features from malware to detection to fragment identification tasks and vice versa. By avoiding the use of extracted file signatures and strings, this study contributes techniques that may be more resistant to obfuscation attacks, lead to enhanced prediction rates for zero-day malware files, and improved forensics on broken fragments where file metadata information is not available. We evaluate our results against recent works and report the highest performing algorithms and combinations of features for each task.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

List of Acronyms and Abbreviations

AUC	area under the curve
FN	false negative
FP	false positive
IDS	intrusion detection system
IM	instant message
IRC	internet relay chat
KNN	k-nearest neighbor
LSVM	linear support Vector Machine
NLP	natural language processing
RATS	remote access tools
ROC	Reciever Operating Curve
Stacked KBL	stacked KNN Bayes LSVM
SVM	support vector machine
TN	true negative
TP	true positive

THIS PAGE INTENTIONALLY LEFT BLANK

Acknowledgments

I am deeply grateful for the tireless guidance and support I received from several individuals, in particular, my advisor Dr. Joel Young. His astute feedback, perceptive and persistent encouragement, patience, inspiring work ethic, and technical support were essential to this work's fruition. Beyond the project, I am in awe of his unabashed dedication to principle, including concern for his students' success. I am humbled, uplifted, and fortunate to be the beneficiary of his mentorship.

I am extremely thankful to Chris Eagle for his patience, impeccably sharp insight, and feedback even on short notice. Thank you for challenging my technical skills and imparting such passion at the same time.

I would also like to thank the many Department of Computer Science faculty and staff who selflessly assisted to make the work possible.

Thank you, Mom, for your wisdom and endlessly patient ear, for your understanding, gentle encouragement, and believing in me. Thank you, Dad, for helping me with years of grade school science projects, for instilling creativity, curiosity, and by example, inspiring me to reach high.

Finally, thank you to my friends and classmates for the tutoring, companionship, and memorable moments, in particular Michael Clement for your coaching and magnanimous brilliance.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1: Introduction

Machine learning algorithms are used in a range of disciplines including computing, software engineering, biology, psychology, and business, and are increasingly in demand for efficient analysis of big data. Machine learning techniques are particularly useful in identifying patterns and extracting points of interest in datasets that would otherwise be unscalable. Furthermore, machine learning can help improve productivity and effective decision making by providing probabilistic predictions about data.

This study leverages machine learning techniques for two important classification tasks in secure computing: malware detection and file fragment identification. While the algorithms we use are standard, the approach to feature generation is novel. Classification is based on information-theoretic descriptions of the byte-level content in each block in a file. By turning our focus away from extracted file content such as signatures and strings, the techniques used in this study may be more robust to obfuscation attacks and provide better classification for similar but previously unseen samples.

Commercial antivirus and academic security researchers have expended significant effort to discover patterns in malware content using data mining techniques. Still, most off-the-shelf solutions are unable to detect encrypted malware [4] as well as malicious code embedded inside a file such as a word document, pdf, or image that otherwise looks benign. Obfuscated malware attacks fly under the radar until the file has been opened or the embedded code has been executed on the victim's system. Most commercial systems rely on a database of signatures such as byte sequences and strings that appear in known malware samples and the tools look for precise signature or rule matches, which are easily evaded. A system is needed that is resistant to obfuscation attacks and that better predicts if a file is malicious even if the content has not been seen before. This study combines a technique proposed by Tabish, Shafiq, and Farooq [4] using a range of information-theoretic features and a technique for fragment identification proposed by Fitzgerald [5] that uses natural language processing (NLP) features on byte-level content for 1-, 2-, 3-, and 4- gram histograms. We observe the effectiveness of various feature combinations and learning algorithms to identify malicious blocks, make predictions about files, and ultimately validate the information-theoretic approach.

Research on the use of machine learning for digital forensics is somewhat less comprehensive than it is in the malware problem space. However, we recognize and leverage a potentially useful similarity: fragment identification also involves classification of binary data. Forensic scientists, like malware analysts, are often faced with more data than can be sorted in an effective amount of time, even with the fastest tools at their disposal. One major, contributing challenge is efficient file carving. In forensic hard-drive analysis, files are often scattered across a device or across multiple devices. When the file is not contiguous, it is difficult to identify and reconstruct the pieces and ultimately, determine whether the file contains criminal evidence. Successes in classifying byte information for malware space classification and its similarities to the underlying tasks involved in identifying file fragments motivates our use of information-theoretic features and standard machine learning algorithms, as with malware, to classy byte level file information. Rather than determining whether a fragment and parent file are malicious, the algorithm predicts the type of the fragment from the set of types provided in training.

1.1 Research Questions

Our major contribution is to validate the use of information-theoretic and NLP features for classifying blocks containing byte-level information. Tabish et al.[4] assert that the information-theoretic method is most effective when all features computed on all 1-,2-,3-, and 4- gram histograms are used with boosted decision tree analysis. Fitzgerald et al.[5] recommend linear SVM with natural language processing features for fragment identification. We evaluate both methods with matching feature sets using the Python Orange machine learning tools. In addition, we extend the work to examine new feature vector combinations. By implementing the methods, we hope to make variables, including type of algorithm, block size, block count, and feature selection more transparent so they may be further optimized in future applications.

1.2 Significant Findings

Our experiments support Tabish's findings [4]: information-theoretic features of byte-level content are useful for malware classification tasks. This study supports the claim by replicating Tabish's procedures, to the extent possible, and showing that additional NLP measures and n-gram distributions do not significantly increase classification performance. In contrast to Tabish, however, we find that bagged k-nearest neighbor learners outperform decision tree algorithms. Additionally, we observe contradictory results to Tabish's study in terms which types of malware are most difficult to classify.

For file fragment tasks, we find, similar to Fitzgerald et al.[5] that NLP measures and n-gram distributions with linear support vector machines, yield promising results. Likewise, larger training and testing files of 4,000 byte blocks (versus 1,000 or 2,000) resulted in higher performance. Expanding on Fitzgerald's work, we show that classification can be improved by including information-theoretic features used in malware detection tasks.

1.3 Document Structure

This thesis focuses on two classifications problems that, we hypothesize, can be resolved using similar techniques. Each section is broken into subsections, first addressing a malware classification problem, then fragment identification. The following summarizes the sections of this thesis:

Chapter 2 provides an overview of research using artificial intelligence, machine learning, and other automated techniques for detecting anomalous content. Section 2.1 surveys research related to static malware detection over the past few decades, while section 2.2 describes recent learning methods for recovering file type information about file fragments.

Chapter 3 summarizes methodologies for obtaining data and running experiments. First, it documents the sources of our training and testing data, including the `VXHeavens` malware dataset, `Govdocs1` document database, and mining methods for obtaining supplemental internet samples. It outlines methods for data selection, provides source statistics, and analyzes potential for noise. It defines the selected learning algorithms, parameters, and feature sets. Finally, Chapter 3 provides mathematical definitions for each information-theoretic and NLP feature, along with our methods for generating feature vectors.

Chapter 4 presents experimental results and a brief comparison to Tabish's and Fitzgerald's conclusions. We sort the results for malware studies by malware type and summarize results for different learning algorithms across types for easy comparison. For each test we observe accuracy, precision, recall, f-score, and the most commonly mis-classified types.

Chapter 5 presents a deeper analysis of the results, potential for error, and future improvements. We address whether the methodologies used in this thesis might be applied in an operational system, whether it be a forensic file carving tool or malware detection suite. We include opportunities for further research and parameter optimization.

CHAPTER 2:
Literature Review

This study focuses on two important classification problems, malware detection and file fragment identification; both areas where machine learning techniques have shown promise. The following chapter addresses relevant applications of machine learning algorithms and feature generation. The review highlights limitations of today's methods, introduces techniques that may prove useful in both malware and forensic disciplines, and motivates a need for advanced predictive solutions that work efficiently in practice.

2.1 Malware Detection

Malware detection technologies, including algorithms used in academic research and enterprise antivirus software like Symantec, are widely available. Detection techniques can be either static or dynamic. Each has noteworthy advantages and disadvantages. Dynamic methods observe the behavioral symptoms of a program as it executes. They are well-suited for classifying potentially malicious behavior, even when a particular attack has not been seen before, but they are not best-suited for on-the-fly prevention. Dynamic analysis usually requires non-trivial computing resources including virtualization, human insight, and time for bootstrapping suspicious versus normal behavior.

Static analysis, by contrast, is the study of non-executing code. While it is more appropriate for live wire applications, it is insufficient against new variants. Signature-based methods are the most common static techniques in academic and commercial solutions. Most antivirus systems perform lookups on a database of known "bad" code sequences to determine whether an incoming file is malicious. Antivirus software and intelligence producers must continuously update signature databases. Advanced techniques, including many of the methods discussed here, use previously seen byte sequences in combination with other heuristic models based on expert rule sets that flag patterns of interest. Both methods can be easily evaded by encoding, obfuscating, or crafting payloads with entirely new signatures or re-crafting malicious files to evade watchlist rules. As with any method that generalizes to new instances, heuristic methods are prone to high false positive rates. This section addresses the limits of relevant, static detection approaches and motivates our continued exploration of information-theoretic features for improved precision and recall.

2.1.1 Heuristics and Inductive Rule-Based Learning

In rule-based learning, experts are employed to engineer a list of features that distinguish malicious code from benign. For example, an analyst might study the Stuxnet worm and identify the unique API calls it makes that would likely be found in similar malicious pieces of code. This method is slow, expensive, and has limited success across the ever-changing malware landscape.

In an early attempt to improve heuristic practice, Schultz et al. [6] profiled 1,001 benign and 3,625 malicious PE executables. They used the *GNU libBFD binutils* library to extract information about the DLLs called in each class. They applied instance-based learning and compared DLL information for an unclassified file to DLL features of known files. The learning algorithm returns the class of the example in the collection which is most similar to the unclassified file based on the presence or absence of DLL calls that are thought to have high information gain.

In addition to using expert introspection, Schultz et al. applied an inductive learning algorithm called RIPPER [7], designed to build a set of rules without prior assumptions about how the known data is similar to the unseen data. RIPPER iteratively constructs a rule set until all positive examples in the training data have been described and then greedily adds rules to exclude negative examples. Finally, it applies a combination of cross-validation and minimum description techniques to generate a reduced set of hypotheses that best approximate the target concept.

Schultz concluded that rule-based methods using DLLs have significant limits. While RIPPER's inductive learning style outperformed the instance-based nearest neighbor method, the optimal ROC point had a relatively high 10% false positive rate and unsatisfactory 75% detection rate. Further considering the human costs of determining appropriate features for rule-based learning, in particular features that are robust to evasion, the technique is not a promising method for improved malware detection.

Other well-researched attempts have been made to reduce the burden on human experts by automating rule-set generation and signature carving. In 1994, researchers from IBM [8] applied speech recognition algorithms to automatically extract telltale byte-sequence signatures. A few years later the researchers applied artificial neural networks [9] to detect variants of boot sector viruses, but were unsuccessful in extending the technique to other types of malware. Even if the attempts to automate rule-generation were successful, the use of rule-based learning provides insufficient detection rates for most operational needs.

2.1.2 Strings and Naïve Bayes

Traditional string-based signature methods rely on unique, previously seen byte sequences instead of attribute rules. The sequences are often generated by an expert or automated methods which carve uniquely identifying strings. Byte patterns are sometimes concatenated to form one long signature and incorporated into a rule table. If an unclassified file matches a signature or rule in the database, it is labeled correspondingly. Despite constant updates, signature databases are never complete with respect to the changing malware landscape.

Schultz's method [6] called `Strings`, provided a significant improvement over traditional byte-sequence pattern matching. Schultz applied a naïve Bayes classifier on ASCII strings obtained using the *GNU strings* utility. The method is probabilistic; it computes the likelihood that a given string is malicious or benign given prior probabilities obtained from the training data.

Extending the *Strings* method, Shultz applied a voting-naïve Bayes algorithm he called `hexdump` [6] which built six naïve Bayes classifiers taking every 6th line of a `hexdump` starting with the first through sixth lines of the file. Multi-naïve Bayes yielded the highest detection rate of 97.76%, with a false positive rate of 6.06% and accuracy of 96.88%. Although it yielded a marginally lower detection rate, single naïve Bayes had a lower false positive rate of 3.8% and higher accuracy of 97.1%. While traditional string-based methods are likely to retain lower false positive rates than statistical methods, they are not best-suited for detecting unseen types. Even worse, they are easily evaded by substituting out the known malicious string or encoding the payload.

2.1.3 N-grams and Byte Level Data Mining

In a series of seminal studies, Li and Stolfo determined that advanced statistical methods could be used with *n-gram distributions* to further improve detection [3], [10], [1]. In one study, they created the `Anagram Packet Analyzer` [11]. `Anagram` is a semi-supervised learning algorithm that filters anomalous network packets based the ratio of unseen high-order byte n-grams ($n > 1$) to total n-grams in the packet, weighted by the number of matches to known malicious n-grams. An unseen n-gram is any n-gram that is not in the training set of known benign files. Anagram tracks known benign and malicious n-grams in separate, space-efficient bloom filters. When applied to whole files rather than packets, a new file is classified based on a model bloom filter built from 5-grams in the training set. In a follow-on experiment, Stolfo [11] parsed files into substructures (e.g. text, tables, macro) and assigned a weight to each n-gram based on the structure it belonged to.

In subsequent investigations, Li and Stolfo [1] applied entropy measures and several classification techniques to both file type and malware domains. They examined the use of n-gram distributions, three different model building methods, and similarity measures to classify file type and benign versus malicious. They hypothesized that n-gram distributions of a file or parts of a file can be used to model the file and discover anomalous binary data. Their file dataset included eight different file types while the malware dataset included infected versions of the same file types. First, they constructed single-centroid models, meaning for each type of file they computed a single model representing that type. Then they computed the Mahalanobis distance of the test file to each centroid model and returned the class of the model with the shortest distance to the test file. Since some file types do not have sufficiently similar distribution to be represented by a single model, the authors also experimented with multi-centroid models. In this case a k-means clustering with Manhattan distance as a similarity metric is applied to compute multiple models for each file type. A test file is compared to all models for each type. Again, the class of the model that is most similar to the test file is returned. In both studies, they evaluated the above methods on truncated file head and tail sections (e.g. the first 10, 50, 200...6000 bytes) of a file and on 1- and 2-gram distributions. By truncating files the authors hoped to determine whether identifying information typically contained in the beginning and end of file is essential for classification, but acknowledged that header information is often damaged or unavailable and would be a poor feature for a practical system. The authors only report detection rates for the head and tail studies (see Table 2.1). They do not report precise AUCs or provide measures of concern if implementing an operational system. Instead, they conclude that the studies are preliminary and suggest increasing further experimentation with increased n-gram size. The results of their file type experiments are more promising (see Table 2.4) and discussed in Section 2.2.

Kolter and Maloof expanded the study of n-gram distributions [2] and evaluated a range of classifiers as shown in Table 2.2. They concluded that several text classifiers are highly

Table 2.1: Li and Stolfo: Malware Detection Using Clustering, after [1, p. 11]

Method	Detection Rate
Head 1000 bytes	87.5%
Head 500 bytes	90.5%
Head 200 bytes	94.5%
Tail 1000 bytes	75%
Tail 500 bytes	80.1%
Tail 200 bytes	72.1%

effective for distinguishing malicious versus benign files. J48 boosted decision trees proved most effective with an AUC of .996 under the ROC approaching 100%. They also conclude that 4-grams are optimal over 1- and 2-grams, and suggest that only the 500 n-grams with the highest information gain are needed as features in any data mining phase.

Kolter and Maloof [2] performed further research to determine whether a malicious file could be classified based on its payload function. Since many executables perform multiple functions (e.g. a payload could be a backdoor and keylogger) the authors chose *one-versus-all* classification, grouping all of the executables with a particular functionality into a class and all others into a non-class. They applied classification to all the models and reported combined positive results. For example, a file could be classified as a backdoor according to one model and a keylogger according to another. Again, boosted J48 decision trees produced the highest AUCs: .88 for mass mail payloads, .87 for backdoor, and .91 for viruses at 95% confidence intervals. Unlike other studies, Kolter and Maloof implemented the system in a real-world, online application and measured the false-positive rates on 291 new, previously unseen executables as shown Table 2.3. Although the performance of the

Table 2.2: Kolter and Maloof: PE Classification with Top 500 N-grams Length 4, after [2, p. 2731]

Method	AUC
Boosted J48	.9958+/-.0024
SVM	.9925+/-.0033
Boosted SVM	.9903+/-.0038
IBk, k=5	.9899+/-.0038
Boosted Naïve Bayes	.9887+/-.0042
J48	.9712+/-.0067
Naÿe Bayes	.9366+/-.0099

system decreased, boosted decision trees yielded less than 10% false positive rates 100% of the time. Overall, the authors conclude that J48 boosted decision trees are highly effective with 4-gram byte distributions for determining maliciousness and, to a moderate extent, payload. However, the results apply only to PE executables and a small subset of possible payload classifications.

2.2 File Type Classification

Similar to malware detection, a number of studies have examined the use of machine learning techniques for file fragment classification.

Li and Stolfo, as cited above [3], used the unigram byte histograms of head and tail sectors for eight file types and applied single centroid, k-nearest neighbor clustering, and centroids with exemplar files to develop models of each type. They achieved superior results for file type identification (see Table 2.4), as opposed to their malware experiments. The highest performing method compared the unigram distributions of test files (a subset consisting of 1 in 5 randomly selected files without replacement) to the remaining training files using Manhattan distance. However, this technique has several limitations for our purposes; it relies on header information that is often not available in forensic practice and provides the type of an entire file. It does not necessarily extend to file fragment identification. It is also largely dependent on file type signatures found in the head and tail, making it easy for attackers to evade.

Calhoun and Coles [12] considered four file types (`jpg`, `bmp`, `gif`, and `pdf`). They used linear discriminant analysis with nine statistical variations of entropy and longest common substring. The algorithm finds linear combinations of features to draw classification lines in pairwise comparisons; e.g. `jpg` vs. `bmp` and `bmp` vs. `gif`. The best discriminant relied on ASCII code frequencies (0x32-0x192), low byte frequencies (<0x32), high byte (>0x32) frequencies, the sum of the four highest byte frequencies, and the standard deviation of the frequencies with 128 bytes of header removed and 1024 byte blocks. It yielded an average accuracy of 88.3% across all four types. They found that different combinations of features and pairs produced better results than other combinations. For example, `jpg` and `bmp` files could be distinguished with 100% accuracy using a combination of all possible features. Interestingly, the best overall discriminant did not include entropy. Veenman [13] also used linear discriminant analysis but expanded the study to 11 file types with Shannon entropy

Table 2.3: Kolter and Maloof: PE Classification of 291 Unseen Samples, after [2, p. 2735]

Method	FP = .01	FP =.05	FP=.1
	Predicted/Actual	Predicted/Actual	Predicted/Actual
Boosted J48	.94 / .86	.99 / .98	1.00 / 1.00
SVM	.82 / .98	.98 / .90	.99 / .93
Boosted SVM	.86 / .56	.98 / .89	.99 / . 92
IBK, k = 5	.90 /.67	.99 / .81	1.00 / .99
Boosted Naïve Bayes	.79 / .55	.94 / .93	.98 /.98
J48	.20 / .34	.97 / .94	.98 / .95
Naïve Bayes	.48 / .28	.57 / .72	.81 / .83

Table 2.4: Li and Stolfo: Fileprint Results, after [3, p. 70]

Experiment	`EXE`	`GIF`	`JPG`	`PDF`	`DOC`	Average
Single-centroid	88.3%	62.7%	84%	68.3%	88.3%	82%
Multi-centroid	88.9%	76.8%	85.7%	92.3%	94.5%	89.5%
Exemplar files	94.1%	93.9%	77.1%	95.3%	98.9%	93.8%

and Kolmogorov complexity as features. He analyzed 4096 byte blocks. His experiments were accurate 45% of the time across the 11 types and found similarly that some types are more easily classified than others.

Axelsson [14] considered a larger set of file fragment types and again found that fragments, especially those with high entropy, were not easily classified even with a different learning method. Axelsson used k-nearest neighbors clustering with compression distance, 10 trials, 10 random test files, and 14 random 512 byte blocks from each, against 3,000 fragments of known types. Of the 28 file types considered, `Java` fragments were most easily classified, though at a disappointing accuracy of 48%. On average, accuracy was 34%.

Conti et al. [15] achieved better results using Shannon entropy, Hamming weight, Chi-square goodness of fit, and mean byte value features grouped by k-nearest neighbor with Euclidean distance. Their study boasts accuracy rates ranging from 88% to 100% depending on type. This is especially impressive given the source of the data which included text, encoded fragments, machine code, and bitmap fragments gathered through various sources or constructed from existing files. The data sources, however, were somewhat contrived and the technique did not perform well on actual file fragments.

During the time that the experiments in this paper were conducted, another study was published that tested combinations of features used in over twenty different file fragment classification studies. It is by far the most comprehensive study to-date. Using a newly constructed (and distributable) dataset with 38 file types, Nicole Beebe et al. achieved 73.4% classification accuracy across all types. They used support vector machine and varied input features to include unigrams and bigrams, complexity, and other byte frequency-based measures [16]. The publication provides a rather comprehensive summary of features used across recent experiments. Beebe concludes that the most performant approach is to use concatenated unigrams and bigrams with linear SVM. An open source tool called `Sceadan` [17], implements their approach. It is available online along with their dataset.

2.3 Summary

This chapter familiarized the reader with a variety of recent machine learning approaches including variations on learning algorithms, input features, and data sources applied to malware classification and fragment identification. The limited success of string and signature based approaches lead us to consider novel methods that employ statistical characteristics of byte content. The following chapters describe our methods for evaluating the information-theoretic based methods asserted by Tabish et al. [4] for malware classification tasks and NLP inspired approach employed by Fitzgerald et al. for file type classification tasks [5].

CHAPTER 3: Methodologies

In this chapter, we describe the methods we used to conduct malware and file fragment machine learning experiments. First, we describe our data sources and attempts to obtain the training and testing sets used by Tabish [4] and Fitzgerald c [5] for replication purposes. We define the features used in our experiments as well as learning algorithms and parameters. This chapter describes the procedures and tools we use to perform malware classification and preliminary file type tests.

3.1 Datasets

This section describes the repositories used in our experiments including their provenance and summary statistics. The following sections explain how data was selected.

For file fragment identification tasks, we required a repository containing a range of file types commonly found on hard drives and in other media of forensic interest. The `govdocs1` corpus by Garfinkel et al. [18] is a rich database with a million files and over 50 different file types. It is the main dataset used in Fitzgerald et al.'s experiments. Unfortunately, we were unable to obtain a precise list of the files used by Fitzgerald [5]. In a best effort attempt to remain as true as possible to Fitzgerald's procedures, we obtained a local version of the `govdocs1` repository and examine the same extensions as those presented in Fitzgerald's 2012 paper [5].

To reproduce information-theoretic feature experiments on malware we needed a set of benign files and malware files of known type, matching those used by Tabish et al. `Govdocs1` provided benign `pdf`, `jpg`, and `doc` files. We obtained `exe`, `zip`, and `mp3` files by mining open-source internet repositories and lab computers. The provenance of these files is described later in this section. It should be noted that the six benign types used by Tabish et al. are common candidates for attackers attempting to obfuscate malicious code in documents, email attachments, and internet downloads. The list of good candidates, of course, is not limited to these six. Like Tabish et al. [4] we utilize the `VXHeavens` [19] malware database, a large sample of malicious files created by a range of novice to professional hackers discovered on the internet over the past several years. We were, again, unable to obtain the precise files used in the original studies, but replicate the experiments as closely as possible.

3.1.1 `VXHeavens` Corpus

Until March, 2012 VXHeavens was a freely available collection maintained by a user named Herm1t at `vx.netlux.org` [19]. Its self-proclaimed goal was to provide expert level information and education about computer viruses. The web server was seized by Ukrainian police and Herm1t was prosecuted in March 2012 to intent the share and sell malicious code. It was one of the few existing, public malware repositories and has been used in a number of academic studies including Tabish et al.'s work [4] [20] - [22] to cite a few. After completing the experiments in this study, `VXHeavens` came back online. It can be accessed at `www.vxheavens.org`.

As recommended by `VXHeavens`'s splash page at the time this study was conducted [23], we sourced our version from The Pirate Bay [24]. The torrent and peer-to-peer download completed on February 21, 2013 delivering 44GB of data and 83,852 files. While there are no official sources to determine provenance for the individual specimens, our review of the files along with `VXHeavens` related blog posts, articles, and other web media suggest that the `VXHeavens` repository is one of the largest collections available, that it contains samples collected over the last decade and likely represents viruses less complex than those produced by nation states and parties with capital resources. Since these experiments, the webpage has been restored and it appears that new owners are crowd sourcing and maintaining an expanded 400GB database. As addressed more thoroughly at the end of this paper, experimenting with new `VXHeavens` samples posted since our 2013 download remains future work. Many of the files in the 44GB `VXHeavens` download used in this study are included in industry alerting databases such as Microsoft's Security Essential's and Symantec's Virus Definition databases [25]. Each file is classified by the its malicious function and the target operating system. A count of each type is shown in Table 3.1 and by the type of operating system the file runs on (e.g. Win32, DOS). The majority, 75,530 files, are Win32 PE format.

3.1.2 Govdocs Digital Corpus

The `govdocs1` digital corpora includes one million files that were obtained by searching `.gov` domains for random combinations of words in the Unix dictionary concatenated with

Table 3.1: Counts for each File Type in the `VXHeavens` Corpus

Label	Count
Trojan	52437
Backdoor	16007
Virus	8304
Worm	1870
Email Worm	1016
Rootkit	991
Exploit	602
Net Worm	582
Hoax	493
Constructor	277
HackTool	204
VirTool	201
P2P Worm	175
Flooder	154
IRC Worm	153
IM Worm	120
Packed	111
DoS	88
SpamTool	17
Spoofer	10
Misc	3

random numbers between 1 and 1 million [18]. The files have been reviewed to determine proper extension, but this is an ongoing and imperfect process. As stated in the `govdocs1` documentation [18], the listed extension is not a precise indicator of actual file type. Table 3.3 shows the estimates provided by the `govdocs1` documentation for the database. We explored alternative methods for identifying the actual file type, but doing this task correctly over a large dataset is beyond the scope of this study. One alternative relies on `libmagic` descriptions instead of the file extension to establish ground truth. Unfortunately, similar challenges exist with both methods. Many files can be classified as more than one type while others are reported as "unknown". For example, `csv` files are essentially `txt` files except that content is separated by strategically placed commas. New versions of Windows Office files are similar to `xlsx`. `Pptx` files are similar to `zip` files. Table 3.2 shows a portion of the output from running the `file` command on `zip` files in `govdocs1`. A variety of types were returned.

Table 3.2: Sample Output of *file* Command on `zip` Files

Filename	'file' Return Value
654117.zip:	Microsoft Excel 2007+
656219.zip:	HTML document
656438.zip:	XML document text
862565.zip:	Microsoft Word 2007+
947671.zip:	Microsoft PowerPoint 2007+

File extension inaccuracies with respect to ground truth are potential sources of noise in training, testing, and the final results.

3.1.3 Supplemental Benign Files

For the malware detection experiments we required a benign dataset that matched Tabish et al.'s set containing six types. Three of these types were not available in `govdocs1`. These exceptions are shown in Table 3.4. We obtained the needed types by mining websites that host open-source collections as well as lab computer drives. In this section we summarize the sources for `exe`, `mp3`, and `zip` files in the benign dataset issued in our malware detection experiments. The dataset is available upon request.

3.1.4 Data Selection

Malware Files

Following Tabish et al. [4], we focus on six malware categories found in the `VXHeavens` collection. We examined samples and describe what is included in each category:

Backdoor

A backdoor is a program that listens for commands over a network connection and gives an attacker remote control of a system. Backdoors typically require client/server components to accomplish the network communication portion. Most backdoors allow an attacker to perform file transfers, acquire passwords, and execute commands. More nefarious versions

Table 3.3: Govdocs Extension Accuracy Estimates

Accuracy Rating	Percentage
High	92.5%
Medium	7.0%
Low	<.25%
None	<.25%

Table 3.4: Source of exe, mp3, and zip Files for Malware Detection

Type	Number	Source	Date
exe	80	http://img.cs.montana.edu/windows/	6/5/13
	77	http://www.fractals.net/public/	6/8/13
	38	http://mirror.thekeelecentre.com/pub/java/	6/10/13
	11	http://sourceforge.com	6/12/13
	27	Student hand-coded in VB, compiled with .NET2003	6/12/13
	138	Windows 7 64bit files and user programs	6/12/13
mp3	1277	ccmixter.com - open source music 6/10/13	6/10/13
zip	284	ftp://download.dosgamesarchive.com/	6/6/13
	10	http://img.cs.montana.edu/windows	6/5/13,
	7	http://sourceforge.net/projects/npp-plugins/files/	6/11/13

include *bots* which automatically cause the infected system to attack other systems. *Zombies*, *bots* and *agents* are typically used to carry out coordinated DDOS attacks. *RATs* or *remote administration tools* allow the attacker to access the system as needed, granting control over the systems devices (webcams, microphones, and speakers).

BACKDOOR EXAMPLE:

> `Rbot` is a family of backdoor that allows the attacker to take control of a victim's machine. `Rbot` connects to an IRC server where it receives commands from attackers. For example, the attacker may run commands that scan the infected computer's network for exploitable windows vulnerabilities, look for file shares with weak passwords, infect other computers on the network, and launch denial of service attacks. [26]

Trojan

Trojans are programs that appear benign but perform covert, malicious activity. Trojans come in variety of forms. Some trojans completely replace an existing program but maintain its functionality, while others simply modify or add additional functions to existing programs. A generic example would be a login program that illegally collects and transmits passwords. Trojans can cause serious technical issues ranging from performance disruption to making the system unusable.

TROJAN EXAMPLE:

> `Trojan.Win32.Buzus` is a program that installs unauthorized files on the infected system, typically a worm, capable of spreading via removable USB drives and performing other unwarranted actions on the system. It is typically distributed as an executable file attached to an e-mail message or downloaded from a compromised website. [27]

Virus

Viruses are designed to self-replicate and distribute themselves to other files, programs, or computers. Viruses are often inserted into a benign carrier. For example, a word document might contain a viral macro. A virus that runs inside or is executed by another application is an *interpreted* virus. Some *compiled* viruses are stand-alone files that can be directly executed by the operating system. Virus impacts range from causing moderately annoying pop-ups to overtly malicious modification, dissemination, or destruction of sensitive data.

VIRUS EXAMPLE:

> `Virus.WWin32.Xorer` a family of file infectors. The virus waits for a certain amount of time to pass between infecting more files. According Microsoft's repository, it encrypts and prepends the virus code to an existing document. It has worm capabilities and is able to drop copies of items in writable drives, as well as rootkit capabilities which allow it to avoid detection. It is a virus that can infect both files and boot sectors. [28]

Worm

Worms are self-replicating, self-propagating, stand-alone programs that can execute without user intervention. Network worms take advantage of network services to infect other systems. One common medium by which a worm spreads is via mass email. The process can overwhelm email services and cause performance issues for infected systems. Worms are also used to call backdoors, perform general denial of service attacks, and aid other types of attacks.

WORM EXAMPLE:

> `Worm.Win32.Autorun` is a family of worms that secretly copies itself into programs and data files. It spreads by copying itself into more files and removable drives every

time the the host program is run. [29]

Constructor

Constructors are malware creation toolkits that allow users to specify settings and automatically assemble new code. They are useful for attackers with little programming experience and for creating polymorphic code [30]. The kits range from simple to very sophisticated in terms of the options and features they provide.

CONSTRUCTOR EXAMPLE:

> `Constructor.Win32.NGVCK.0_40` is a variant of `Program:Win32/Advdown.A` that downloads malware from http://www.paragon-software.com/ and mainly targets computers in Nepal, Moldova, Togo, Somalia, and Vietnam. [31]

Miscellaneous

The `misc` category includes exploits, flooders, hacktools, virtools, and hoaxes. Exploits, hacktools, and virtools are programs that take advantage of vulnerabilities, such as causing buffer overflow so the attacker gains execution control. Flooders run mass email, instant messaging, and SMS attacks. Hoaxes are non-malicious alerts and programs that cause damage through social engineering. Hoaxes can cause a backlog of complaints to IT departments or trick users into making changes to security settings that are less secure.

Selection: Malware and Benign Files for Training and Testing

Tabish et al. built six separate classifiers, one for each category of malware using 50 random files of the specific type of malware (see Table 3.5) and a total of 50 random benign `doc`, `jpg`, `exe`, `pdf`, `zip`, and `mp3` files. They do not specify how many of each benign type were represented. Tabish et al. claim to test their models on all remaining files which would seem to include the entire `VXHeavens` database and approximately 295 files of each benign type. Since Tabish et al. were unable to provide their precise dataset upon request, we chose to test the types equally. Ultimately, we collected a subset of 300 files for each of the following types: virus, trojan, worm, constructor, backdoor, misc, `exe`, `zip`, `mp3`, `doc`, `pdf`, and `jpg`. Following the cited method, we created six training sets, one for each malware type. Each training set is comprised of blocks from 100 files including 50 randomly selected malware files of the type in question and 50 total benign files (8-9 of each type). Table 3.6 summarizes the composition of the six training models, one for each malware type, used in our classification experiments. Likewise, we create six test sets each containing the remaining 250 malware files of the particular type and 250 benign files (41-42 of each benign type), so that files used in training are excluded from testing. A summary of the test files is given in Table 3.7. It is important that no training files are used in testing since one of our goals is to verify whether the Tabish method is effective on unseen files.

File Fragment Identification

Like Fitzgerald et al. we source from the `govdocs1` corpus to obtain a variety of file types. There are 50 different extensions represented in the repository and Fitzgerald et al. study 24 of them. This study examines the 23 extensions highlighted in Table 3.8. The frequency of types across the `govdocs1` corpus is not uniform. Following Fitzgerald, our study excludes uncommon extensions that are not of interest as well as those where the sample is not large enough to ensure separate training and testing. One of such types is `.zip` files. Although Fitzgerald et al. cite 10 `zip` files in the `govdocs1` database, we discovered that these files are not actually `zip` files. Therefore the `govdocs1` repository does not have sufficient representation of `zip` files and we have decided to exclude them from all our trials. In the fashion of Fitzgerald's work, this study assumes that extension is ground truth for a file's type even though this method can produce inaccuracies.

From the list of almost 1,000,000 files in the the `govdocs1` repository, we randomly selected sets of 1,000, 2,000, and 4,000 blocks of 1,204 bytes from each of the 23 types excluding file header and footer blocks. We exclude header and footer blocks because these sections often contain descriptive flags that make identification too easy. It is preferable to represent a variety of different sample files for each type so that the model is not overtrained to a particular file. We also ensure that the training and testing datasets are comprised of blocks from mutually exclusive files even though other papers do not appear

Table 3.5: Training Subset: Malware Files

Type	Files	Average Size	Blocks
Backdoor	50	447119	21832
Trojan	50	229042	11184
Misc	50	189015	9229
Constructor	50	392253	19153
Virus	50	95821	4679
Worm	50	219779	10731

Table 3.6: Six Training Models: One for Each Malware Type and All Benign Types

Model	Malware Files	Benign Files
Backdoor	50 backdoor subset	50 benign subset
Constructor	50 constructor subset	50 benign subset
Misc	50 misc subset	50 benign subset
Trojan	50 trojan subset	50 benign subset
Virus	50 virus subset	50 benign subset
Worm	50 worm subset	50 benign subset

Table 3.7: Six Test Sets: One for Each Malware Type and All Benign Types

Model	Malware Files	Benign Files
Backdoor	250 backdoor subset	250 benign subset
Constructor	250 constructor subset	250 benign subset
Misc	250 misc subset	250 benign subset
Trojan	250 trojan subset	250 benign subset
Virus	250 virus subset	250 benign subset
Worm	250 worm subset	250 benign subset

concerned about this. We applied the following methodology to ensure sufficient representation of each type from a variety of files and that there is no overlap in training and testing data.

First, we randomly selected 1,000, 2,000 and 4,000 files of each type assuming random ordering of the file list. We then randomly selected one 1,024 byte block from each file without replacement of blocks until there were 1,000, 2,000 or 4,000 blocks per file type. For types where there were less than 1,000, 2,000 or 4,000 distinct files, some files are revisited and another random block is selected from the file. We call this an *limited* dataset: each block comes from a distinct file. Put differently, there is a 1:1 block-to-file ratio. The types of files represented in the limited dataset are shown in Table 3.10. The *unlimited* dataset includes subsets of 1,000, 2,000 and 4,000 blocks per type but excludes types without enough files to achieve a 1:1 block-to-file ratio. See Table 3.9 for a summary. For the unlimited sets that included all types, we divided each 1,000, 2,000 and 4,000 block pool into 2 pools to obtain a 9:1 training-to-testing ratio. To prevent any overlap between training and testing files in the limited set, we ignore any blocks remaining at the end of a file after building the training set. We start building the test set from a new, unused file in the pool. Note that, for these types, there is not a perfect 9:1 ratio. In the limited sets, files

Table 3.8: `Govdocs1` File Types Used in Fragment Experiments Highlighted

Extension	Count of files	MB
`pdf`	232794	127492
`html`	191407	11222
`**jpg`	109278	35970
`*text`	83805	50406
`doc`	80648	30099
`xls`	66599	29041
`ppt`	50257	122918
`xml`	41994	8405
`gif`	36301	2920
`ps`	22129	27668
`csv`	18396	3347
`gz`	13870	8651
`log`	10241	4204
`unknown`	8188	4456
`eps`	5465	3082
`png`	4125	1079
`swf`	3691	1853
`pps`	1629	3629
`kml`	995	149
`kmz`	949	226
`hlp`	660	5
`sql`	632	226
`dwf`	474	42
`java`	323	11
`*txt`	286	9
`pptx`	219	562
`tmp`	196	15
`docx`	169	32
`ttf`	104	1
`js`	92	2
`pub`	76	1
`bmp`	75	31
`xbm`	51	1
`xlsx`	46	6
`jar`	34	2
`zip`	27	1
`wp`	17	2
`sys`	8	0
`dll`	7	0
`exe`	5	0
`exported`	5	0
`**jpeg`	3	0
`tif`	3	0
`chp`	2	0
`data`	1	0
`pst`	1	0
`squeak`	1	12
Total	986278	477641
*types can be collapsed		

represented in training were necessarily distinct from the files represented in testing. This guarantee allowed us to use Orange's [32] built in cross-validation. The experiments are described further in the next section. In total, our process yielded six datasets as shown in Table 3.9 and Table 3.10.

3.2 Feature Generation

This section defines the features extracted from malware and benign files, including the so-called information-theoretic features used by Tabish [4] and so-called NLP features used by Fitzgerald [5]. Although their naming is somewhat specious, we have chosen to adopt the authors' original terminology for simplicity throughout this paper. The features discussed in this section were extracted for each block in the testing and training datasets. We

Table 3.9: File Type Classification on Unlimited Dataset: Blocks Come From Distinct Files

Blocks per type	Total types	Included types (distinct original files)
1000	16	`jpg gz png ppt doc pdf txt html xml xls gif ps csv swf pps rtf`
2000	14	`jpg gz png ppt doc pdf txt html xml xls gif ps csv swf`
4000	13	`jpg gz png ppt doc pdf txt html xml xls gif ps csv`

Table 3.10: File Type Classification Limited Dataset: Blocks Can Come From Repeat Files

Blocks per type	Total types	Under-represented types (repeat original files)
1000	23 (all)	`pptx docx xlsx sql java tex bmp`
2000	23 (all)	`pptx docx xlsx rtf pps sql java tex bmp`
4000	23 (all)	`swf pptx docx xlsx rtf pps sql java tex bmp`

performed experiments using different combinations of selected features to determine the features with highest accuracy and lowest false positive rate.

3.2.1 Information-Theoretic Features

Information theory provides mathematical methods for quantifying information such as the limits on data compression, reliable transmission, and data storage [33]. Note that information theory is not concerned with the semantic meaning or function of the data. Instead, it describes numeric qualities of the data. Table 3.11 summarizes the features we compute and the previous studies that motivated us to include them.

Some of the features are based on common information-theoretic measures computer science, biology, ecology, and other statistical sciences. A subset of these features are more commonly used for language processing tasks. Several so-called NLP features are used in Fitzgerald's study, and therefore, in our experiments as well. In total, we include 17 distinct information-theoretic features. The features used in Tabish's malware study [4] are computed for 1-, 2-, 3-, and 4-gram frequency distributions. Following Fitzgerald's example [5], these include NLP features on per block unigram frequencies (count of byte values 0-256). We experiment with combinations proposed by both authors and various combinations of features. Experiments are described in Section 3.4.

Following Tabish et al.'s work [4], we adopt their terminology to reference the statistical features they present, but we recognize that it is somewhat an abuse of notation to say we

Table 3.11: List of Features

Feature	Computed On frequencies	Source
Kolmogorov complexity	Sliding byte window (unigram)	Fitzgerald et al.
Hamming weight	Sliding byte window (unigram)	Fitzgerald et al.
contiguity	Sliding byte window (unigram)	Fitzgerald et al.
longest repeating seq	Sliding byte window (unigram)	Fitzgerald et al.
Simpson's index	1-,2-,3-,4-gram	Tabish et al.
Shannon entropy	1-,2-,3-,4-gram	Tabish et al.
angular separation	1-,2-,3-,4-gram	Tabish et al.
Bray-Curtis divergence	1-,2-,3-,4-gram	Tabish et al.
Canberra distance	1-,2-,3-,4-gram	Tabish et al.
Chebyshev distance	1-,2-,3-,4-gram	Tabish et al.
correlation coefficient	1-,2-,3-,4-gram	Tabish et al.
Itakura-Saito distance	1-,2-,3-,4-gram	Tabish et al.
Jensen-Shannon distance	1-,2-,3-,4-gram	Tabish et al.
Kulback-Leibler divergence	1-,2-,3-,4-gram	Tabish et al.
Manhattan distance	1-,2-,3-,4-gram	Tabish et al.
Minkowski distance	1-,2-,3-,4-gram	Tabish et al.
variation	1-,2-,3-,4-gram	Tabish et al.

apply "information-theoretic formulas". We simply take advantage of the fact that these measures produce a numeric description that may prove useful in identifying and classifying the information encoded in each block. Typically, information-theoretic distance and divergence metrics are used to observe the difference between two points or input vectors, for example, the difference between a two vectors representing prior and actual probability distributions. This study applies information-theoretic formulas on two input vectors, but unlike typical applications, our vectors are derived from the same frequency distribution.

We denote the first vector as X_i. It represents the left end of each n-gram frequency distribution such that $X_i = (X_0, X_1, \ldots, X_{n-2})$, $n = 2^{8x}$, and x is the $n - gramsize$ in bytes. For example, if x =1 byte, the first frequency distribution counts the occurrences of n-gram values in range $(0, 1, 2, \ldots, 254)$ for each block. The second vector X_{i+1} is the same distribution shifted by 1. It represents the right end of the frequency curve such that $X_{i+1} = (X_1, X_2, \ldots, X_{n-1}), n = 2^{8x}$. If $x = 1$ byte again, X_{i+1} counts the frequency of n-grams whose values range from $(1, 2, 3, \ldots, 255)$. We reflect the fact that the feature generation methods are somewhat contrived with respect to their typical information-theoretic uses, by representing the feature formulas with prime (’) and the input vectors as X_i and X_{i+1} rather than p and q.

Ultimately, the goal of applying features based on information theory is to produce a quantitative profile of the binary information contained in each file. Even if those measurements do not have clear mathematical meaning, they may have be useful descriptors and provide an identifiable signal. The selected features are defined as follows:

Minkowski Distance
The Minkowski distance gives the m-order distance or the power mean of the component-wise differences between two points or vectors. It is a generalization of Manhattan distance and Euclidean distance.

$$Minkowski(p,q) = \left(\sum_{i=1}^{n} |p_i - q_i|^m \right)^{1/m} \tag{3.1}$$

$$Minkowski'(X_i, X_{i+1}) = \left(\sum_{i=0}^{n-1} |X_i - X_{i+1}|^m \right)^{1/m} \tag{3.2}$$

Chebyshev Distance
The Chebyshev distance is the maximum distance between two vectors along any coordinate system. In other words, it is the limit of the Minkowski distance as m goes to infinity.

$$Chebyshev(p,q) = \lim_{k\to\infty} \left(\sum_{i=1}^{n} |p_i - q_i|^k \right)^{1/k} = \max_i(|p_i - q_i|) \tag{3.3}$$

$$Chebyshev'(X_i, X_{i+1}) = \max_i(|X_i - X_{i+1}|) \tag{3.4}$$

Manhattan Distance
The Manhattan distance is the geometric, right angle distance between two points in vector space given a Cartesian coordinate system. It is computed by taking the sum of line segment lengths between points along coordinate axes. It is an instance of the Minkowski distance where m=1.

$$Manhattan(p,q) = \sum_{i=1}^{n} |p_i - q_i| \tag{3.5}$$

$$Manhattan'(X_i, X_{i+1}) = \sum_{i=0}^{n-1} |X_i - X_{i+1}| \tag{3.6}$$

Canberra Distance

Canberra distance is a weighted version of the Manhattan distance. Typically it is used to find the normalized distance between two points (p,q) in vector space. For example, in intrusion detection Canberra distance might be used to quantify how similar a suspect file is to a predetermined, normal model.

$$Canberra(p,q) = \sum_{i=1}^{n} \frac{|p_i - q_i|}{|p_i| + |q_i|} \tag{3.7}$$

$$Canberra'(X_i, X_{i+1}) = \sum_{i=0}^{n-1} \frac{|X_i - X_{i+1}|}{|X_i| + |X_{i+1}|} \tag{3.8}$$

Bray Curtis Distance

The Bray Curtis distance is a measure of dissimilarity between two frequency distributions. It is commonly used in ecology to measure the difference between compositions of species at two sites. It can also be considered a normalized Manhattan distance.

$$BrayCurtis(p,q) = \frac{\sum_{i=1}^{n} |p_i - q_i|}{\sum_{i=1}^{n} (p_i + q_i)} \tag{3.9}$$

$$BrayCurtis'(X_i, X_{i+1}) = \frac{\sum_{i=1}^{n} |X_i - X_{i+1}|}{\sum_{i=1}^{n} (X_i + X_{i+1})} \tag{3.10}$$

Angular Separation

Angular separation is a similarity measure based on the cosine of the angle between two vectors. High angular separation means that the input vectors are similar.

$$AngularSep(p,q) = \frac{\sum_{i=0}^{n-1}(p_i)(q_i)}{\sqrt{\sum_{i=0}^{n-1}(p_i)^2 \sum_{i=0}^{n-1}(q_i)^2}} \tag{3.11}$$

$$AngularSep'(X_i, X_{i+1}) = \frac{\sum_{i=0}^{n-1}(X_i)(X_{i+1})}{\sqrt{\sum_{i=0}^{n-1}(X_i)^2 \sum_{i=0}^{n-1}(X_{i+1})^2}} \tag{3.12}$$

Correlation Coefficient

Correlation coefficient is the separation between vectors centered around the mean of the vector coordinates. A high correlation coefficient implies similarity.

$$CorrCoeff(p,q) = \frac{\sum_{i=1}^{n}(p_i - \bar{p})(q_i - \bar{q})}{\sqrt{\sum_{i=1}^{n}(p_i - \bar{p})^2 \sum_{i=1}^{n}(q_i - \bar{q})^2}} \tag{3.13}$$

$$CorrCoef'(X_i, X_{i+1}) = \frac{\sum_{i=0}^{n-1}(X_i - \bar{X}_i)(X_{i+1} - \bar{X}_{i+1})}{\sqrt{\sum_{i=0}^{n-1}(X_i - \bar{X}_i)^2 \sum_{i=0}^{n-1}(X_{i+1} - \bar{X}_{i+1})^2}} \tag{3.14}$$

Shannon Entropy

Shannon entropy is a measure of how much information is contained in a message or how much information is missing if a symbol in a message is unknown. Entropy quantifies the dispersal of a distribution and the uncertainty of a random variable.

$$Entropy(X) = -\sum_{i=1}^{n} p(x_i)\log_2 p(x_i) \tag{3.15}$$

where $p(x_i)$ is the probability of x_i.

$$Entropy'(X) = -\sum_{i=0}^{n} p(x_i)\log_2 p(x_i) \tag{3.16}$$

where $p(x_i)$ is the probability of the ith n-gram in a block.

Total Variation

In probability, total variation is the largest possible distance between two distributions for a given event. It is related to the Kolmogorov-Smirnov test [34] which measures the distance between an empirical and cumulative distribution for a sample.

$$Variation(p,q) = \frac{1}{2}\sum_{x}|p(x) - q(x)| \tag{3.17}$$

$$Variation'(X_i, X_{i+1}) = \frac{1}{2}\sum_{i}|X_i - X_{i+1}| \tag{3.18}$$

Kullback-Leibler Divergence

Kullback-Leibler divergence [35] measures how much information is lost by trying to approximate one distribution with another. It is often used to encode and quantify how far a model or theory is from a true distribution and is the basis for information gain calculations used in decision tree induction.

$$KullLeib(p \parallel q) = \sum_{i=1}^{n} \log\left(\frac{p(i)}{q(i)}\right) p(i) \tag{3.19}$$

$$KullLeib'(X_i \parallel X_{i+1}) = \sum_{i=1}^{n-1} \log\left(\frac{X_i}{X_{i+1}}\right) X_i \tag{3.20}$$

Jensen-Shannon Divergence

Jensen-Shannon divergence [36] is a symmetric, smoothed version of Kullback-Leibler divergence. It measures similarity between probability distributions.

$$JenShan(p \parallel q) = \frac{1}{2}K(p \parallel m) + \frac{1}{2}K(q \parallel m) \tag{3.21}$$

where $m = \frac{1}{2}(p+q)$.

$$JenShan'(X_i \parallel X_{i+1}) = \frac{1}{2}D(X_i \parallel M) + \frac{1}{2}D(X_{i+1} \parallel M) \tag{3.22}$$

where $M = \frac{1}{2}(X_i + X_{i+1})$.

Itakura-Saito Distance

Itakura-Saito distance [37] is an example of Bregman divergence, a generalized Euclidean Distance that maintains certain convexity, non-negativity, and duality properties.

$$ItakSaito(p,q) = \sum_{i=1}^{n} \left(\frac{p(i)}{q(i)} - \log \frac{p(i)}{q(i)} - 1 \right) \tag{3.23}$$

$$ItakSaito'(X_i, X_{i+1}) = \sum_{i=1}^{n} \left(\frac{X_i}{X_{i+1}} - \log \frac{X_i}{X_{i+1}} - 1 \right) \tag{3.24}$$

Simpson's Index

Simpson's index [38] is commonly used in ecology to determine the probability that two specimens selected at random from an ecosystem will belong to the same class. The variable n represents the number of organisms of a particular class and N is the total count of all organisms. For binary file information, n is the count of each possible n-gram value and N is the total number of n-grams in a block.

$$Simpsons = \frac{\sum_{i=0}^{n} n(n-1)}{N(N-1)} \tag{3.25}$$

$$Simpsons' = \frac{\sum_{i=0}^{k} n_k(n_k-1)}{N(N-1)} \tag{3.26}$$

where n_k is the frequency of the kth value and N is the number of n-grams in a block.

3.2.2 Natural Language Processing Features

The following features are used by Fitzgerald et al. [5] to distinguish file fragments. Following Fitzgerald et al., we apply them in our file fragment identification experiments. We also include them in some of our malware detection trials with promising results. While the measures described in this section are common in language processing tasks, they do not strictly belong to that domain of problems. We adopt Fitzgerald's notation throughout this paper.

Contiguity

Contiguity measures the average distance between byte sequences. Essentially, it is a heuristic that can potentially determine if there is a general, repeating pattern to the data. In this study we only compute the contiguity between consecutive 1-byte n-grams, but in future research it might prove useful to explore other pattern lengths. When computing different n-gram lengths, it will also be necessary to consider the various offsets at which a pattern might start. For feature generation, we define contiguity as:

$$Contiguity = \sum_{i=0}^{k-1} \frac{|n_k - n_{k+1}|}{k} \tag{3.27}$$

where k is the block size in bytes.

Hamming Weight

Hamming weight is computed by summing the 1s and dividing by the total number of bits in a binary string. For a given string, it is the same as the Hamming distance from the all-zero string of equal length.

$$HammingWeight = \sum_{i=0}^{k} \frac{n_k}{k} \tag{3.28}$$

where k is the block size in bytes.

Longest Repeating Sequence Length

Some files contain long strings of a repeated byte sequence. Empty document files, for example, contain long strings of 0s. This measure is computed as the length of the longest repetition of any single byte value. It is also possible to take the string made up of any single repeating n-gram value. For simplicity, we observed only the longest string with respect to bytes.

$$LongRep = \max_i length(s_i) \tag{3.29}$$

where s_i is the longest string consisting only of bytes with value i.

Kolmogorov Complexity

The Kolmogorov complexity, also known as algorithmic complexity, is the length of the shortest descriptor or language needed to produce a string. It can be approximated using compression algorithms like *bzip* and *gzip*. We apply Python's *zlib* compression function to obtain an estimate.

$$Kolmogorov(s) = |d(s)| \tag{3.30}$$

where $d(s)$ is the shortest descriptor of s.

3.3 Learning Algorithms

This section is a review of the learning algorithms used in each experiment. Each algorithm was applied to the training sets to obtain a classifier. Each classifier was then evaluated on the test sets previously described.

3.3.1 k-Nearest Neighbor (KNN)

KNN is an instance-based learning method that classifies objects by taking a majority vote among its neighbors. The training examples are represented as vectors in a multidimensional space and their ground truth classes are known. The user defines a positive constant integer k which is the number of neighbors that will be allowed to vote. A distance such as Euclidean distance or Hamming distance determines the k closest neighbors in the feature space to the object in question. The new object is labeled with the class that is most common amongst the neighbors. This is disadvantageous when the data is skewed. For example, a class may dominate the feature space if there is a disproportionate number of examples of that class to other classes.

Several parameter optimizations have been proposed to improve the performance of KNN learners. One method is to weight neighbors by their distance to the new point. It may also be productive to explore the optimal value for k. This is important because a k that is too large will not be able to distinguish the classes. Another method is to vary the distance metric used to determine nearest neighbors. Scaling and dimension reduction are often utilized to reduce noise and eliminate irrelevant features. Some studies use genetic algorithms in preprocessing to determine the optimal parameter values.

3.3.2 Decision Trees

Decision trees are an inductive method that create a list of parameters from a set of features and training data, and classify a new file based on those parameters. Following its name, combinations of parameters can be represented as branches on a tree. Classes are represented as tree leaves. There are a number of decision tree algorithms for generating rules from the training data. The algorithms vary in terms of what measure is used to split data, whether the predicted class is continuous or discrete, how over-fitting is handled, and whether missing values are permitted. The most common are C4.5, its ancestor ID3 [39], and the Weka implementation of J48.

For this study, decision trees are implemented with Python Orange's *orange.TreeLearner* module, which uses a top-down induction method similar to C4.5. By default, Orange [40] uses the information gain ratio (information gain divided by the entropy of the feature's value) to determine parameters for splitting data at each node [40]. Information gain is also known as the Kullback-Leibler divergence. It is a normalized measure of the difference in entropy, or how much more information is available given the true value of a variable in a probability distribution. At the beginning of the induction process all features are considered possible candidates for rule generation. Nodes are generated recursively, taking the feature with the highest information gain that has not already been used at another node. Although the Python Orange implementation is similar to the Weka J48 implementation used in the Tabish study [4], it is not precisely the same.

3.3.3 Naïve Bayes

Naïve Bayes classifiers are probabilistic models in which the given probability of features in training independently influence the probability of the object's class. Despite the unrealistic assumption that features are independent, Naïve Bayes classifiers are a useful solution to a host of real classification tasks.

This study applies the Bayes classifier included in Python Orange [41]. According to documentation, Orange implements a version of Bayes's algorithm as defined by Tom Mitchell [42]. It is a standard implementation and similar to the the Weka implementation used in Tabish et al.'s work. Essentially, the classifier returns the class with the highest likelihood based on prior probabilities of each feature chosen to model the instance:

$$classify(f_1,\ldots,f_n) = \operatorname*{argmax}_c \, p(c)\prod_{i=1}^{n} p(f_i \mid i) \tag{3.31}$$

3.3.4 Support Vector Machines

Support vector machines are a classification and pattern matching method that build a representation of training data with p features in p-dimensional space [43]. In a linear SVM, the space is split by a $(p-1)$-dimensional hyperplane that maximizes the gap between data points belonging to two different classes. A support vector machine evaluates new examples by first mapping them into the partitioned space and then returning a classification based on where the new data point falls in relation to the hyperplane [43]. More formally, a linear support vector hyperplane is represented as

$$P = \{(x_i,y_i) \mid x_i \in \mathbb{R}^p,\, y_i \in \{-1,1\}\}_{i=1}^{n} \tag{3.32}$$

where sub_{y_i} is either 1 or -1 and represents the class of x_i, and x_i is a p-dimensional vector. The hyperplane creates maximum distance between the points where $y_i = 1$ and those where $y_i = -1$.

Non-linear SVMs apply a kernel so that the hyperplane is mapped into a transformed a formulaic space; they are useful when the data cannot be linearly separated. Common kernel transformations include Gaussian radial basis, polynomial, and hyperbolic tangent functions.

This study utilizes several of the SVM classifiers available in Python Orange's *LibSVM* library [44]. The *orange.SVMLearner* module allows the user to specify a kernel parameter. In this experiment we utilize the linear, polynomial, radial, and sigmoid kernels. The Orange implementations are equivalent to those used in Tabish's study [4].

3.3.5 Stacked Learners

As addressed in the discussion of our experimentation design, we utilized several Python Orange libraries to perform classification trials. Unlike Weka and other readily available machine learning libraries, Python Orange affords the ability to test a combination of learners. According to available documentation, the method infers a meta-classifier from class probability estimates on preliminary cross-validation data [45]. This method can produce meta-classifiers that have greater prediction accuracy than the individual classifiers in its substructure. We apply a Stacked classifier that includes KNN, naïve Bayes, and linear SVM sub-learners It is referenced throughout this paper as a "Stacked KBL" classifier.

3.4 Experiments

This section describes the procedures used in our malware detection and fragment identification experiments. We apply the classifiers described in the previous section to three feature combinations; a set that includes the information-theoretic features from the Tabish study [4], a set that includes the features from the Fitzgerald study [5], and a set that combines features from both studies.

3.4.1 Malware Classification

A major goal of this thesis is to validate Tabish et al.'s [4] work which recommends using all information-theoretic features on 1024 byte blocks. Following their methods, we ran three experiments with a preliminary block size of 1024 bytes.

We ran ran three malware classification sets, each with a different feature vector. In "Tabish" trials we apply the the 13 information-theoretic features on 1-, 2-, 3-, and 4-grams from Tabish's study [4], shown in Table 3.11. This yields a total of 52 features.

In the second trial, we apply five Fitzgerald features (Kolmogorov complexity, Hamming weight, contiguity, longest repeating string, and Shannon entropy) [5] to unigram distributions. We also include 255 unigram counts (values 0 to 255) making a total of 261 features. From this point forward we refer to any trial using this feature vector as a "Fitzgerald" experiment.

Finally, we combine features from both papers in what we henceforth call "Combined" experiments. This feature vector includes the 52 features from "Tabish" experiments, 256 unigram counts and 4 "Fitzgerald" features, 5 features minus Shannon entropy which is already included in the "Tabish" set. "Combined" experiments apply 312 features.

In total we perform 36 classification experiments: six learning algorithms: (KNN, bagged KNN, linear SVM, Stacked KBL, regression trees, and naïve Bayes) are applied to each of the six malware models. After models for each classifier and malware category are built, we extract features on each test block and classify it as either *malicious* or *benign*. When the model has made a prediction for every block in a test file, majority voting is used to determine the class of the entire file. The threshold in our study and Tabish et al. [4] is 50%. For instance, if there are 100 blocks in a file and a model classifies 49 blocks as *benign* and 51 blocks as *malicious*, the predicted class for the entire file is *malicious*. In the edge case where the votes are equal, the file is labeled *malicious*. The classifiers are evaluated on the set of benign files and malware files not used in training. For example, a tree learner is applied to a training set that includes backdoors and benign files. The trees are tested on leftover benign files and backdoor files, but not on worm, virus, trojan, constructor, or miscellaneous files. Figure 3.1 summarizes the overall experiment design from the feature generation phase to block classification and file classification by majority vote.

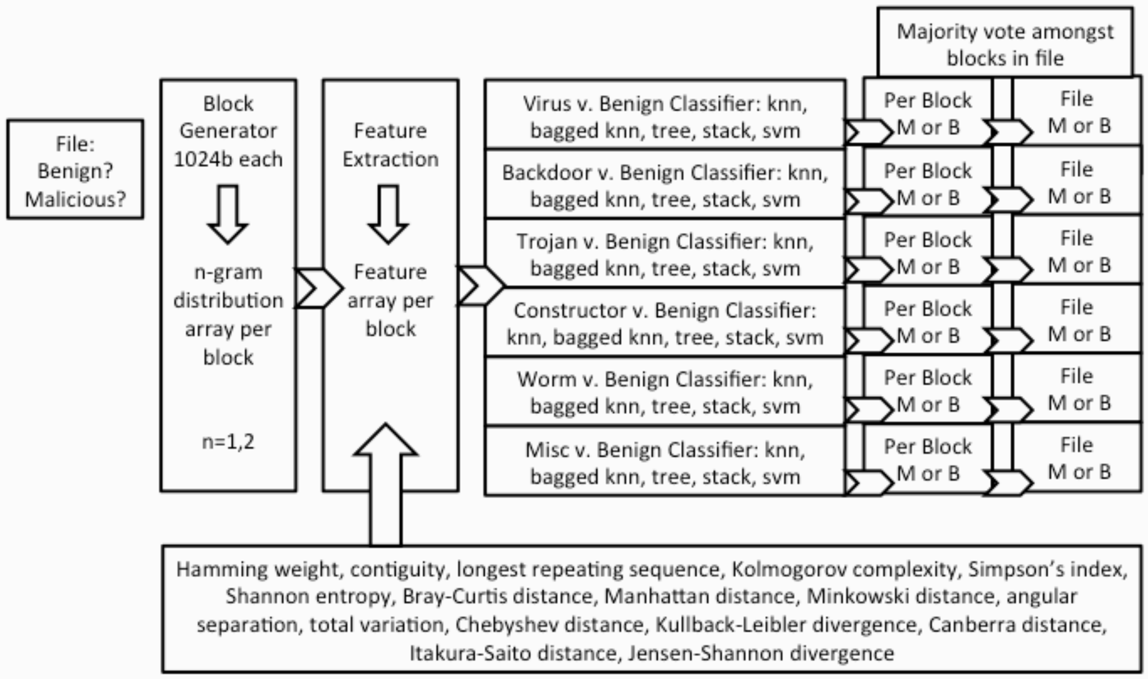
3.4.2 File Fragment Identification

The following trials are designed to validate the use of NLP features and n-gram frequencies for fragment classification and evaluate the potential usefulness of information-theoretic features combined with NLP features. We classify a limited set of file types compared to the 23 types presented in Fitzgerald's study [5]. Our pilot study only includes the the types with enough files from our data sources to take be able to take one or fewer blocks from each file. Like Fitzgerald, we experiment with the number of blocks used in training (1,000, 2,000 or 4,000).

Expanding on Fitzgerald's work [5] with linear SVM, we test five additional learning algorithms from the Python Orange library on each dataset (1,000, 2,000 and 4,000 blocks). The six Orange learners used are: KNN, bagged KNN, Stacked KBL, regression trees, naïve Bayes, and linear SVM. The classifiers produced a file type prediction for each of the blocks in the test set. Ultimately, we apply Orange's 10-fold cross-validation tool to obtain accuracy, precision, recall, and f-score.

Since the goal is to validate or disqualify the need for further research on information-theoretic features in file type identification tasks, we only present results for "Combined" feature vector experiments. Similar to our second malware classification trial, the vector includes 312 features, 52 from "Tabish" and 260 from "Fitzgerald". Although limited in scope, our trials produced promising results; details are provided in Chapter 4. Chapter 5, addresses variations that are not included but are worthy of future research.

Figure 3.1: Malware Vs. Benign File Experiment Design


File: Benign? Malicious?
Block Generator 1024b each
n-gram distribution array per block
n=1,2
Feature Extraction
Feature array per block
Virus v. Benign Classifier: knn, bagged knn, tree, stack, svm
Backdoor v. Benign Classifier: knn, bagged knn, tree, stack, svm
Trojan v. Benign Classifier: knn, bagged knn, tree, stack, svm
Constructor v. Benign Classifier: knn, bagged knn, tree, stack, svm
Worm v. Benign Classifier: knn, bagged knn, tree, stack, svm
Misc v. Benign Classifier: knn, bagged knn, tree, stack, svm
Per Block M or B
File M or B
Majority vote amongst blocks in file
Hamming weight, contiguity, longest repeating sequence, Kolmogorov complexity, Simpson's index, Shannon entropy, Bray-Curtis distance, Manhattan distance, Minkowski distance, angular separation, total variation, Chebyshev distance, Kullback-Leibler divergence, Canberra distance, Itakura-Saito distance, Jensen-Shannon divergence

3.5 Summary

This chapter described our data sources, learning algorithms, methods of feature selection, and experiment processes for malware and file type classification. Chapter 4 presents the results of our experiments and an analysis of our results compared to Tabish's [4] and Fitzgerald's [5] findings. We evaluate the highest performing combinations of features, learning algorithm, and block size as a result of our methods and their potential for further investigation.

CHAPTER 4:
Results

4.1 Introduction

In Chapter 3, we outlined the methods used to calculate features for the files we want to classify. This included computing the same information-theoretic features that Tabish et al. [4] used to identify malware files, as well as NLP and unigram distributions that Fitzgerald et al. [5] used to predict a file's type from incomplete fragments. We also outlined where we obtained our training data, how we performed machine learning, and used voting to obtain a prediction. In this section we present the results including accuracy, precision, recall, and f-score. This section also compares our results to [4] and [5] where relevant.

4.2 Malware Detection

This section presents the results for our malware classification experiments. It includes side-by-side comparisons of the top performing learner on on the various types of malware and addresses results for the the three different input vectors. Results are grouped into sections by malware type. The majority are shown in table format with accuracy, recall, precision, and f-score moving across the rows.

4.2.1 Tabish Features

Below are the results for our first phase of experiments, which used 52 "Tabish" features to identify benign files versus six types of malware. In general, we find that accuracy rates are similar, but not quite as strong, as those reported by Tabish et al. [4]. Overall the highest performing learner is bagged KNN (Table 4.1). Except on constructor and worm files, bagged KNN yielded the highest f-scores.

Trojan Detection

Of the six classifiers, the highest performing classifier for trojans versus benign files was bagged KNN with a coefficient of $k = 3$. It resulted in .91 accuracy, although it did not yield particularly impressive false negative and false positive rates. Bagged KNN had .63 recall and .62 precision. Although the Python Orange stacked KBL method produced fewer false negatives with a recall (.71), KNN maintained the highest f-score (.75). See Table 4.2.

Table 4.1: Tabish Features: Top Learner Results by Malware Type

Class	Learner	Accuracy	Recall	Precision	F-score
Miscellaneous	Bagged KNN	0.93	0.7	0.72	0.82
Virus	Bagged KNN	0.93	0.44	0.98	0.75
Constructor	Stack KBL	0.9	0.86	0.54	0.78
Trojan	Bagged KNN	0.91	0.63	0.62	0.75
Backdoor	Bagged KNN	0.87	0.88	0.48	0.75
Worm	Bagged KNN	0.9	0.51	0.6	0.7

Table 4.2: Tabish Features: Trojan Detection

Dataset↓	Accuracy	Recall	Precision	F-score
KNN (k=3)	0.90	0.66	0.58	0.62
KNN (BAGGED, k=3)	0.91	0.63	0.62	0.75
LINEAR SVM	0.86	0.38	0.43	0.56
STACK K B L	0.88	0.71	0.5	0.72
REGRESSION TREE	0.86	0.33	0.44	0.54
NAIVE BAYES	0.69	0.51	0.2	0.43

Backdoor Detection
Similar to trojan detection, bagged KNN outperformed the other 5 classifiers in terms of accuracy (.87) and f-score (.75). In this case, bagged KNN also outperformed stacked KBL methods on all fronts, including recall (.88) and precision (.48). See Table 4.3.

Virus Detection
Again, bagged KNN performed best out of the 6 classifiers. With an accuracy of .93 and f-score of .75, it is also one of the highest performing Tabish models over all the malware types. Linear SVM produced similar accuracy rates (.91), but did not perform as well in terms of recall (.33) and precision (.88). See Table 4.4.

Table 4.3: Tabish Features: Backdoor Detection

Dataset↓	Accuracy	Recall	Precision	F-score
KNN (k=3)	0.85	0.90	0.46	0.60
KNN (BAGGED, k=3)	0.87	0.88	0.48	0.75
LINEAR SVM	0.78	0.4	0.25	0.46
STACK K B L	0.83	0.84	0.41	0.68
REGRESSION TREE	0.84	0.45	0.34	0.55
NAIVE BAYES	0.7	0.38	0.17	0.37

Table 4.4: Tabish Features: Virus Detection

Dataset↓	Accuracy	Recall	Precision	F-score
KNN (k=3)	0.92	0.46	0.82	0.59
KNN (BAGGED, k=3)	0.93	0.44	0.98	0.75
LINEAR SVM	0.91	0.33	0.8	0.62
STACK K B L	0.78	0.94	0.35	0.64
REGRESSION TREE	0.9	0.24	0.95	0.55
NAIVE BAYES	0.77	0.91	0.35	0.63

Worm Detection
Unlike KNN classifiers for trojan, backdoor, and virus detection, KNN did not outperform other methods in terms of f-score (.70) on worm files. Stacked KBL performed better in terms of precision (.46) and recall (.78), achieving an f-score of .71. Bagged KNN did, however, produce the most accurate results (.90) of the six classifiers. Since the accuracy is significantly higher for bagged KNN versus stacked KBL (.78), we consider bagged KNN the top performer on worms. See Table 4.5.

Constructor Detection
Bagged KNN performed well for constructor identification, but not as well as the stacked KBL method. Bagged KNN produced an accuracy of .88 and f-score of .74; while Stacked SVM achieved accuracy of .9 and an f-score of .78. This is the only class for which we do not consider bagged KNN to be the highest performer. See Table 4.6.

Miscellaneous Malware Detection
Bagged KNN performed significantly better than other methods for malware that did not fall into one of the previously discussed classes. In fact bagged KNN achieved an accuracy of .93, matching bagged KNN results for what Tabish [4] suggest is the easiest malware to classify in terms of accuracy. Bagged KNN also outperformed other learning methods

Table 4.5: Tabish Features: Worm Detection

Dataset↓	Accuracy	Recall	Precision	F-score
KNN (k=3)	0.90	0.59	0.59	0.59
KNN (BAGGED, k=3)	0.9	0.51	0.6	0.7
LINEAR SVM	0.87	0.3	0.45	0.52
STACK K B L	0.86	0.78	0.46	0.71
REGRESSION TREE	0.88	0.06	0.8	0.21
NAIVE BAYES	0.69	0.7	0.24	0.49

Table 4.6: Tabish Features: Constructor Detection

Dataset↓	Accuracy	Recall	Precision	F-score
KNN (k=3)	0.87	0.89	0.46	0.61
KNN (BAGGED, k=3)	0.88	0.88	0.48	0.74
LINEAR SVM	0.75	0.84	0.29	0.57
STACK K B L	0.9	0.86	0.54	0.78
REGRESSION TREE	0.69	0.84	0.25	0.52
NAIVE BAYES	0.72	0.66	0.24	0.49

in terms of precision (.72), recall (.7), and f-score (.82). This is also the highest f-score produced in any of the Tabish trials and malware types. See Table 4.7.

4.2.2 Fitzgerald Features

The following are results of experiments where we applied 261 NLP and unigram distribution frequencies that Fitzgerald used to identify file fragment types [5]. For these trials we applied the features and again use six learning algorithms to predict whether a file is malicious or benign. Similar to our results after applying Tabish features, bagged KNN generally performed best; for the majority of trials. It achieved the highest accuracy and f-score. See Table 4.8.

Backdoor Detection

An exception to all other classification experiments using Fitzgerald's feature set, the highest performing method for backdoor detection was KNN not bagged KNN. Although its accuracy (.85) is just below that of stacked KBL (.86), the recall for KNN is a relatively high .90 and its f-score (.60) outperformed most other methods. It is worth noting that linear SVM also produced decently high results and an f-score of .61 that is just above the f-score for KNN, but it did not fair as well for accuracy (.8) For all remaining mal-

Table 4.7: Tabish Features: Miscellaneous Detection

Dataset↓	Accuracy	Recall	Precision	F-score
KNN (k=3)	0.90	0.66	0.61	0.63
KNN (BAGGED, k=3)	0.93	0.7	0.72	0.82
LINEAR SVM	0.84	0.44	0.38	0.57
STACK K B L	0.88	0.75	0.53	0.74
REGRESSION TREE	0.77	0.72	0.32	0.58
NAIVE BAYES	0.74	0.79	0.3	0.57

Table 4.8: Fitzgerald Features: Top Learner Results by Malware Type

Miscellaneous	Bagged KNN	0.93	0.67	0.76	0.82
Constructor	Bagged KNN	0.92	0.85	0.6	0.81
Virus	Bagged KNN	0.92	0.58	0.76	0.78
Trojan	Bagged KNN	0.91	0.63	0.64	0.76
Worm	Bagged KNN	0.9	0.54	0.61	0.71
Backdoor	KNN	0.85	0.90	0.46	0.60

ware classes, the highest performing learner was bagged KNN. However, bagged KNN performed surprisingly poorly for backdoor detection with an accuracy of .33 and f-score of .31. See Table 4.9.

Constructor Detection

KNN, bagged KNN, and stacked KBL algorithms all performed well on constructor malware files. Bagged KNN achieved the highest accuracy (.92) and f-score (.81). Its recall (.85) and precision (.6) are slightly higher that of KNN (.89 and .46) and stacked KBL (.84 and .52). The results are not as strong for linear SVM, regression trees, and naïve Bayes. Still, it is worth noting that all accuracy, precision, and recall scores are above .70 and have an f-score above .58. See Table 4.10

Miscellaneous Malware Detection

Similar to results for constructor files, all algorithms produced noteworthy results on miscellaneous files that did not fall into one of the other malware categories. Again, bagged KNN produced the highest accuracy (.93) and f-score (.82). Stacked KBL produced reasonably close accuracy (.92) and an f-score of (.81) as shown in Table 4.11.

Table 4.9: Fitzgerald Features: Backdoor Detection

Dataset↓	Accuracy	Recall	Precision	F-score
KNN (k=3)	0.85	0.90	0.46	0.60
KNN (BAGGED, k=3)	0.33	0.96	0.15	0.31
LINEAR SVM	0.8	0.72	0.35	0.61
STACK K B L	0.86	0.35	0.43	0.54
REGRESSION TREE	0.54	0.97	0.21	0.45
NAIVE BAYES	0.71	0.39	0.18	0.38

Table 4.10: Fitzgerald Features: Constructor Detection

Dataset↓	Accuracy	Recall	Precision	F-score
KNN (k=3)	0.87	0.89	0.46	0.61
KNN (BAGGED, k=3)	0.92	0.85	0.6	0.81
LINEAR SVM	0.76	0.83	0.3	0.58
STACK K B L	0.89	0.84	0.52	0.76
REGRESSION TREE	0.77	0.76	0.3	0.58
NAIVE BAYES	0.75	0.68	0.26	0.52

Table 4.11: Fitzgerald Features: Miscellaneous Detection

Dataset↓	Accuracy	Recall	Precision	F-score
KNN (k=3)	0.90	0.66	0.61	0.63
LINEAR SVM	0.87	0.61	0.48	0.68
KNN (BAGGED, k=3)	0.93	0.67	0.76	0.82
STACK K B L	0.92	0.7	0.69	0.81
REGRESSION TREE	0.81	0.67	0.36	0.61
NAIVE BAYES	0.76	0.76	0.31	0.57

Trojan Detection

Bagged KNN, again, produced the best results. It achieved the highest accuracy (.91) and f-score (.76). While regression trees achieved high precision (.79), they achieved poor recall (.2). KNN and stacked KBL achieved similar, notable accuracy rates (.9). See Table 4.12.

Virus Detection

Like the other malware experiments using Fitzgerald's unigram distribution and NLP features, bagged KNN achieved best results for predicting virus files. While several algorithms scored .92 in accuracy (KNN, Linear SVM, and bagged KNN), bagged KNN achieved the highest f-score of .78. Linear SVM yielded similar though slightly weaker results with an f-score of .75. The regression tree learner produced a high precision of .87, but did not

Table 4.12: Fitzgerald Features: Trojan Detection

Dataset↓	Accuracy	Recall	Precision	F-score
KNN (k=3)	0.9	0.66	0.58	0.62
LINEAR SVM	0.88	0.46	0.52	0.64
KNN (BAGGED, k=3)	0.91	0.63	0.64	0.76
STACK K B L	0.9	0.59	0.6	0.73
REGRESSION TREE	0.89	0.2	0.79	0.48
NAIVE BAYES	0.72	0.52	0.23	0.46

perform as well on recall (.34), and thus, overall f-score (.64). Table 4.13 displays the results.

Worm Detection

In contrast to the results of bagged KNN with Tabish features, bagged KNN yielded better results than other classifiers with the Fitzgerald feature model. Although KNN and regression trees also achieved an accuracy of .9, bagged KNN produced the experiment high f-score of .71. This f-score is significantly higher than that of other learners. KNN, the learner that produced the next highest f-score yielded only .59 for precision, recall, and f-score. Stacked KBL yielded an extremely high recall of .98, but suffered the trade off in precision (.23). See Table 4.14i.

4.2.3 Combined Features: Tabish Information-Theoretic and Fitzgerald NLP Features

This section discusses the results of malware classification experiments using a total of 312 information-theoretic features, unigram distributions, and NLP, the union of features recommend by Tabish [4] and [5]. Overall, combining all the features did not produce higher results than using just one of the features sets. As we saw with Tabish and Fitzgerald [4] [5] feature vectors in the previous two sections,features, bagged KNN performed better than other learners for the majority of malware classes. See Table 4.15. for a summary of best to worst performance using combined features.

Backdoor Detection

Scores for combined features on backdoor files were low compared with scores on other malware types using the combined features, as well as compared to experiments applying just one feature set. Also, this is the only experiment where linear SVM showed strong

Table 4.13: Fitzgerald Features: Virus Detection

Dataset↓	Accuracy	Recall	Precision	F-score
KNN (k=3)	0.92	0.46	0.82	0.59
LINEAR SVM	0.92	0.53	0.74	0.75
KNN (BAGGED, k=3)	0.92	0.58	0.76	0.78
STACK K B L	0.78	0.94	0.36	0.65
REGRESSION TREE	0.91	0.34	0.87	0.64
NAIVE BAYES	0.78	0.94	0.36	0.64

Table 4.14: Fitzgerald Features: Worm Detection

Dataset↓	Accuracy	Recall	Precision	F-score
KNN (k=3)	0.9	0.59	0.59	0.59
LINEAR SVM	0.88	0.38	0.52	0.6
KNN (BAGGED, k=3)	0.9	0.54	0.61	0.71
STACK K B L	0.58	0.98	0.23	0.48
REGRESSION TREE	0.9	0.2	0.84	0.49
NAIVE BAYES	0.71	0.41	0.19	0.39

Table 4.15: Combined Features: Top Learner Results by Malware Type

Miscellaneous	Bagged KNN	0.93	0.68	0.72	0.81
Virus	Bagged KNN	0.92	0.53	0.75	0.75
Worm	Bagged KNN	0.9	0.53	0.62	0.71
Trojan	Bagged KNN	0.89	0.52	0.58	0.69
Backdoor	Stack KBL	0.87	0.66	0.48	0.69
Constructor	Stack KBL	0.87	0.66	0.48	0.69

performance against other learners. Linear SVM matched stacked KBL with the highest f-scores (.69). Still, stacked KBL was able to produce a slightly higher accuracy (.87) compared to linear SVM (.84). KNN and bagged KNN produced slightly weaker results. See Table 4.16.

Constructor Detection

Both bagged KNN and stacked KBL performed well for identifying malware constructor files, yet stacked KBL produced slightly stronger results. Stacked KBL yielded an accuracy of .93 and f-score of .84. Bagged KNN yielded an accuracy of .91 and f-score of .79. These were significantly better than the remaining learners and seem to perform best on constructor files regardless of features selected. See Table 4.17 for results using combined

Table 4.16: Combined Features: Backdoor Detection

Dataset↓	Accuracy	Recall	Precision	F-score
KNN (k=3)	0.85	0.90	0.46	0.60
LINEAR SVM	0.84	0.82	0.43	0.69
KNN (BAGGED, k=3)	0.75	0.92	0.33	0.61
STACK K B L	0.87	0.66	0.48	0.69
REGRESSION TREE	0.52	0.99	0.21	0.44
NAIVE BAYES	0.71	0.39	0.18	0.38

features, Table 4.6 for Tabish results and Table 4.10 for Fitzgerald results.

Miscellaneous Malware Detection
Bagged KNN performed best for classifying malware that did not fall into any of the other malware classes using a combined feature set, as with other feature sets. It achieved a high accuracy (.93) and the high f-score compared to other learners in the group. In general though, all learners achieved relatively high accuracy; all were above .89. See Table 4.18 shows the results.

Trojan Detection
Bagged KNN and stacked KBL yielded similarly promising results for detecting trojan files. Bagged KNN produced a slightly higher accuracy of .89 compared to stacked KBL with an accuracy of .88. Both learners yielded f-scores of .69. Overall, none of the learners performed particularly well in terms of precision and recall on trojans using the combined feature set. Trojan prediction was better under our ‘Tabish”model [4] (Table 4.2) and “Fitzgerald” model [5] (Table 4.12). See Table 4.19 for combined feature results.

Virus Detection
Bagged KNN produced the strongest results for virus detection using combined features. Although all learners yielded high accuracies of .92 (except naïve Bayes), Bagged KNN yielded the highest f-score (.75). It is worth noting that recall (.92) was much stronger than precision (.51) for Bagged KNN. Similarly, stacked KBL had strong recall (.94), but weak precision (.36). Interestingly, linear SVM and regression trees produced similar f-scores of .74 and .72 respectively, but precision was significantly stronger than recall. The regression tree, for example, yielded a precision of .92 while recall was only .42. See Table 4.20.

Table 4.17: Combined Features: Constructor Detection

Dataset↓	Accuracy	Recall	Precision	F-score
KNN (k=3)	0.87	0.89	0.46	0.61
LINEAR SVM	0.81	0.81	0.35	0.63
KNN (BAGGED, k=3)	0.91	0.85	0.57	0.79
STACK K B L	0.93	0.85	0.66	0.84
REGRESSION TREE	0.74	0.86	0.29	0.57
NAIVE BAYES	0.74	0.68	0.26	0.52

Table 4.18: Combined Features: Miscellaneous Detection

Dataset↓	Accuracy	Recall	Precision	F-score
KNN (k=3)	0.90	0.66	0.61	0.63
LINEAR SVM	0.89	0.62	0.53	0.71
KNN (BAGGED, k=3)	0.93	0.68	0.72	0.81
STACK K B L	0.92	0.73	0.66	0.8
REGRESSION TREE	0.84	0.65	0.4	0.64
NAIVE BAYES	0.75	0.76	0.31	0.57

Table 4.19: Combined Features: Trojan Detection

Dataset↓	Accuracy	Recall	Precision	F-score
KNN (k=3)	0.90	0.66	0.58	0.62
LINEAR SVM	0.88	0.45	0.51	0.63
KNN (BAGGED, k=3)	0.89	0.52	0.58	0.69
STACK K B L	0.88	0.58	0.53	0.69
REGRESSION TREE	0.89	0.2	0.78	0.48
NAIVE BAYES	0.72	0.51	0.22	0.45

Table 4.20: Combined Features: Virus Detection

Dataset↓	Accuracy	Recall	Precision	F-score
KNN (k=3)	0.92	0.46	0.82	0.59
LINEAR SVM	0.92	0.51	0.73	0.74
KNN (BAGGED, k=3)	0.92	0.53	0.75	0.75
STACK K B L	0.78	0.94	0.36	0.64
REGRESSION TREE	0.92	0.42	0.92	0.72
NAIVE BAYES	0.78	0.94	0.35	0.64

Worm Detection
The strongest algorithm for worm detection was bagged KNN. It yielded the highest accuracy (.9) and f-score (.71) in the experiment. Although KNN and regression trees achieved the same accuracy (.9) they performed poorly in terms of precision an recall, yielding f-scores of .59 and .52 respectively. Also, worth noting, stacked KBL achieved very strong recall (.98), but this was at the cost of precision (.23). See Table 4.21.

4.2.4 Malware Detection Result Summary

Regardless of the feature set used, we found worms and backdoors hardest to classify. Accuracy and f-scores were generally lowest across trials for these two types. On the other hand, miscellaneous files were easiest to predict. Viruses were, not surprisingly also

Table 4.21: Combined Features: Worm Detection

Dataset↓	Accuracy	Recall	Precision	F-score
KNN (k=3)	0.9	0.59	0.59	0.59
LINEAR SVM	0.88	0.38	0.51	0.59
KNN (BAGGED, k=3)	0.9	0.53	0.62	0.71
STACK K B L	0.63	0.98	0.25	0.52
REGRESSION TREE	0.9	0.2	0.86	0.48
NAIVE BAYES	0.71	0.38	0.18	0.38

relatively easy to classify compared to other classifiers. The latter two types had the highest accuracies and f-scores across trials, regardless of feature set. Additionally, we find that bagged KNN is the most consistent learner; it yielded the highest accuracy and f-score results across the majority of malware and feature vector variations.

4.3 File Fragment Identification

This section presents the results of preliminary trials in which we validate some of Fitzgerald's work using n-gram distributions and linear SVM and show that information-theoretic features may aid Fitzgerald's approach.

We performed several pre-trial tests using Python Orange's 10 fold cross validation tool to explore the use of information- theoretic and NLP features for file type classification. The section is divided by same three input vector variations we used for malware detection; Tabish [4] features only, Fitzgerald [5] features only, and combined features. Similar to Fitzgerald [5] we performed these trials on 1,000, 2,000, and 4,000 blocks, each consisting of 1,024 bytes. Note that, although our dataset included all the file types tested by Fitzgerald et al. we only present the results for file types where we obtained a sufficiently large dataset to generate 4,000 block chunks without reusing blocks from the same files.

Summary of File Fragment Trial Results

Out of four learners applied to the file fragment datasets, the highest performance was achieved using linear SVM (See Table 4.22. We also find that classification is improves with larger block count. Accuracy was higher using 4,000 blocks versus 1,000 or 2,000 blocks (See Table 4.23). It is unclear whether continuing to increase the number of blocks would further improve results. However, we note that it becomes increasingly difficult to find training files large enough to accommodate trials with larger block counts without reusing data.

Similar to Fitzgerald [5], we find that high entropy file types are more difficult to classify. Image files like png and compressed gz files, where repeated information is eliminated to reduce storage (thus increasing entropy), had the lowest prediction accuracies. Table 4.24 shows the results of using linear SVM on 4,000 blocks of each file type. Like Fitzgerald, we found that this combination of learner and size yielded the highest accuracies. After combining Fitzgerald's features with information-theoretic features we obtained significantly

Table 4.22: Results: LSVM and Combined Features by Block Size

Blocks	Accuracy	Precision	Recall	f-score	AUC
1000	0.588	0.639	0.567	0.601	0.891
2000	0.619	0.674	0.577	0.622	0.901
4000	0.659	0.685	0.581	0.629	0.916

Table 4.23: Results: 4000 Blocks and Combined Features by Learner

Blocks	Unigrams	Learner	Accuracy
4000	y	NAÏVE BAYES	0.4533
4000	y	REGRESSION TREE	0.5033
4000	y	KNN (k=3)	0.5877
4000	y	LINEAR SVM	0.6458
4000	n	NAÏVE BAYES	0.3612
4000	n	REGRESSION TREE	0.4171
4000	n	KNN (k=3)	0.5177
4000	n	LINEAR SVM	0.504

Table 4.24: Results: 4000 Blocks, LSVM, and Combined Features by File Type

File Type	Accuracy	Precision	Recall	F-Score
`png`	0.906	0.408	0.5	0.449
`gz`	0.907	0.401	0.425	0.413
`jps`	0.911	0.451	0.71	0.552
`gif`	0.937	0.574	0.712	0.636
`pdf`	0.93	0.582	0.309	0.404
`ppt`	0.934	0.663	0.301	0.414
`html`	0.967	0.822	0.725	0.77
`xml`	0.971	0.823	0.796	0.809
`csv`	0.993	0.937	0.968	0.953
`txt`	0.955	0.657	0.857	0.744
`ps`	0.987	0.904	0.93	0.917
`doc`	0.947	0.685	0.581	0.629
`xls`	0.973	0.879	0.75	0.81

higher accuracy scores compared to those reported by Fitzgerald.

4.4 Results Summary

Overall, our outcomes support the methods employed by Tabish [4] and Fitzgerald [5], with some differences in terms of highest performing feature set and learning algorithm. Although we did not perform malware detection experiments using the same Weka J48 implementation of decision trees as Tabish et al. [4] we generally find that KNN outperforms tree based methods. Going beyond Tabish's work, we find that enhanced meta-classifiers such as stacked ensembles packaged with Python Orange, further improve classification results. We were surprised to find that certain classes in our dataset, such as miscellaneous files, were easy to classify by contrast to Tabish's results [4].

Based on preliminary trials, we find that linear SVM and large block sizes are most effective for file fragment classification, as suggested by Fitzgerald [5]. We were unable to present results for all file types tested in Fitzgerald's study, but generally find that high entropy files are more difficult to classify. Beyond Fitzgerald's results, we find that including additional information- theoretic features improves file fragment classification.

The next chapter analyzes our results and compares them to the historical studies which influenced our work in greater detail. Further, it addresses limitations and provides recommendations for future research.

CHAPTER 5: Conclusions And Future Work

5.1 Conclusions

The experiments described in this paper examined the use of non-signature based features as suggested by Tabish [4] and Fitzgerald [5] We observe their effectiveness in two important problem spaces: malware classification and file fragment identification. Although we were unable to obtain the original researchers' malware test data, we followed the methods defined by Tabish et al. [4] and confirmed some of their conclusions. We also examined additional non-signature features, including the NLP features that were used by Fitzgerald et al. [5] to identify file types from fragments. We applied different combinations of these feature sets as well as different learning algorithms used by both research groups. Finally, we performed preliminary studies using Tabish's features for file fragment identification and noticed increased learner performance with feature sets augmented by Tabish's information-theoretic features.

5.1.1 Malware Detection

Our malware experiments support Tabish et al.'s claim [4] that malware files are different from benign files at a byte level and that this information can be used to gain some statistical intelligence about the type of a malicious file. Using 13 information-theoretic features on 1-,2-,3-, and 4-gram byte sequences, all learners achieved accuracy rates between .87 and .93 and f-scores between .7 and .82, suggesting that there is an discernible difference between malware and benign files. Unlike, Tabish et al., however, we find the bagged KNN generally performed better than the recommended J48 decision trees. We should note though, that bagged KNN is not the most efficient learner in terms of speed; it is particularly slow compared to other learners. Our experiments also resulted in lower performance for viruses and higher performance for miscellaneous and constructor files than expected based on Tabish et al.'s results. Tabish [4] concludes that virus files were easiest to classify, generating the largest area under a ROC curve, and miscellaneous files were most difficult to classify. See Table 5.1 for a side-by-side comparison. Our experiments resulted in similarly high virus file and miscellaneous file classification outcome. Surprisingly, bagged KNN achieved an even higher f-score for miscellaneous files than for virus files.

Table 5.1: Malware Classification Difficulty: Our Results vs. Tabish

Most Difficult↓	Tabish Results	Our Results
	Miscellaneous	Backdoor
	Constructor	Worm
	Backdoor	Trojan
	Trojan	Virus
	Worm	Constructor
	Virus	Miscellaneous

In addition to supporting Tabish's findings [4], this study shows that using additional so called NLP features, including those applied by Fitzgerald et al. [5], does not significantly increase classification performance on different types of malware files. Using Fitzgerald features alone and augmenting Tabish features with Fitzgerald features yielded similarly high accuracy rates and f-Scores, as opposed to using Tabish features alone. The range of maximum accuracies for Tabish trials is .87 to .93, as compared to .85 to .93 for Fitzgerald features, and .87 to .93 for combined features across six classes. F-score ranges have similar distribution as well: .7 to .82 for Tabish trials, .60 to .82 for Fitzgerald trials, and .69 to .81 for combined features.

5.1.2 File Fragment Identification

Preliminary trials using both Tabishi [4] and Fitzgerald [5] features, LSVM, and 4,000 file fragments per file type yielded promising results with significantly higher accuracy compared to Fitzgerald's results [5]. Even for high entropy files that were more difficult to classify, our pretrial results were significantly higher, with all accuracies exceeding .9. In Fitzgerald's experiments [5], LSVM yielded prediction accuracies below .9 for several of the types included in our pretrials: `png`, `doc`, `txt`, `gz`, `zip`, `ppt`, and `jpg`. Our pretrials achieved overall accuracy of .65 across 13 file types, while Fitzgerald achieved an accuracy of .34 across 24 types. The results of our trials motivate further experimentation using n-gram distributions with information-theoretic features, particularly on an expanded list of file types. Even when the number of classes is expanded to include additional file types, recently published research provides evidence that a combined feature set is useful for enhanced fragment classification. Most recently, Beebe et al. [16] achieved overall prediction accuracy of .73 across 30 different file types using a combination of features from twenty different classification experiments. Beebe's work includes several information-theoretic features not used in Fitzgerald's work [5] and several additional features not used in this study.

Similar to Fitzgerald [5] and Beebe et al. [16], we find that LSVM is the highest performing learner for file fragment identification. LSVM performed better than KNN, naïve Bayes, and decision trees. We also find that including unigram distributions and 4,000 fragments per type (versus 1,000 or 2,000) increased the performance of LSVM on fragment classification. As discussed in the following section, additional experiments are needed to examine the effect of manipulating the number of file fragments, combinations of selected n-gram, and information-theoretic features.

5.2 Future Work

The following section recommends variations on the experiments described in this paper for follow-on study. It also addresses implementation issues and areas requiring further research if the techniques are to be incorporated into useful tools for information security analysts.

5.2.1 Parameter Optimization

This thesis led to the development of a framework that facilitates building trained models and testing various parameters. The studies presented here focus on recommendations provided by Tabish [4] and Fitzgerald [5]. This work examines a few additional combinations of Tabish's and Fitzgerald's feature sets. However, there are many "levers" that can be varied in future studies that may achieve even more optimized performance and results.

1. **Block Size:** The block size is an important feature because it defines the minimum size for malware or a fragment that can ibe detected in a model. Decreasing the block size, of course, increases the amount of time and computation required to build a model and test a file, but has potential to further improve results in both the malware and fragment domains.
2. **N-gram Size:** The maximum n-gram size is set to four bytes in this work and recent studies. In the future, our extraction and feature models can be optimized for better memory management and processor efficiency to allow computation on longer n-grams. Similar to decreasing the block size, increasing n-gram length may allow the the model to detect sequences that were not detected in this study. The n-gram size

is an upper bound on the size of an instruction or “word” with high information gain.

3. **Feature Set:** This study applies information-theoretic features gathered from a number of recent byte-level classification proposals. We observe that results were best when we included a combination of information-theoretic, Natural Language Processing, and n-gram measures in the feature set. This superficially suggests that there is a positive relation between number of features and results. However, we also observe that the difference is insignificant for malware classification tasks, but significant for file fragment identification. Further work is needed to observe the effect of more features such as larger n-gram counts as well as more focused sets that parse out features with the highest information gain.

   Beebe et al.’s work [16], which examines a variety of unigram, bigram, and information-theoretic input vectors, suggests that future work focus not on increasing the number of features, but on finding the optimal set of types of features. They find that performance degrades, due to over-fitting, not necessarily when more features are used, but when several different types of features are used. Beebe et al.’s work [16] also suggests that future experiments utilize unigram and bigram concatenation, as this was found to be superior to other input vectors when a large number of file and data types were involved. Future studies would also feature support vector machines with a linear kernel, as suggested by Beebe et al. [16].

4. **Classes:** Another limitation of this research and related studies, is that learning is performed on a finite set of malware and file type classes. This study examines six malware categories; a list which could be expanded to include bot code, web injects, rootkits, and many other common malware types. For file fragments, our goal was only to examine the potential of combined unigram and information-theoretic features (not to build a comprehensive classifier), but we only examine present 13 file types. Like Beebe et al. [16] whose study examines 30 types and is the most comprehensive to date, we recommend that future work expand the research to include common data types such as `elf`, `sql`, `rtf`, `swf`, `wav`, `mov`, and `wma`.

5. **Training Data:** We use widely available datasets for training file fragment and malware models. As discussed in Chapter 3, there are a number of limitations to these sets. In future file identification studies, we recommend using data that is labeled with `libmagic` or other methods that are comprehensive and accurate. For malware

studies, a repository containing more recent samples from diverse sources is needed. While the methods proposed in this thesis produce good results when the test data is not used in training, the test data comes from the same repositories. Future studies, should seek a dataset with samples from a diverse time period, including current samples for training and testing. This is especially important considering the speed of malware evolution and how quickly training data can become outdated.

6. **Learner Parameters:** Most programmatic learners require user specified parameters. In file fragment identification for example, we instruct the support vector machine to split the vector space with a linear kernel. There are many other types of kernels, including custom kernels, that may prove even more effective. In trials, we experimented with a radial kernel. The results are not officially presented, though we did not find that it significantly outperformed linear SVM. Our study affirms the conclusions presented in other related literature as well. We find that decision trees perform better than regular Bayesian approaches, KNN, and linear SVM for malware detection. However, there are many settings for decision trees including pruning (pruning = 2 in our study) and the measure or algorithm used to split the tree (information gain ratio in our study). Future experiments are needed to determine optimal learning algorithm parameters.
7. **Boosting:** Boosting methods can improve supervised learners by reducing bias and combining weak learners into a strong learner. As our results, affirm, byte-level classification across multiple types is difficult. We observe that stack and boosting methods significantly improve the performance for malware detection. In the future, it would be interesting to apply boosting techniques to SVM for file fragment identification tasks as well and experiment with stacked ensemble combinations.
8. **Systematic Method and Voting Schemes:** By performing classification on blocks of byte-level information, our framework provides some insight into the potentially interesting sections of a file.

   With file fragments, it may be productive to look at confidence intervals for file class given a block's predicted class. Other work can be done to determine the accuracy of the voting model used in malware detection for file fragmentation to predict the extension for a file of unknown type. How many blocks of a file are needed to achieve highly confident overall file predictions? In addition, it would be interesting to experiment with classifying smaller, more focused type categories. A framework that classifies files into groups might be particularly useful when it is known *a priori* that a file belongs to a limited set of possibilities. For example, if we know the fragment comes from an image, does prediction improve for a model trained just on `gif`, `jpg`, `png`, and other image files? Would using sub-classifiers or sub-categories like "image" be combined to create an even stronger learner? These questions are a natural extension of our work, but even more importantly, are synergistic with realities of forensic practice for file fragment identification.

There are also a number of potentially useful modifications that might be made to the malware framework used in this study. Naturally, the voting threshold used to determine whether a file is malicious can be adjusted. In future studies, we recommend examining the ROC curve generated by adjusting the threshold above and below our current setting of 50%. In the future we intend on testing new malicious and benign files from sources other than the `VXHeavens` data and observing the effectiveness of the classifiers in a live intrusion detection system (IDS). Testing the framework on new files is needed to further address whether the model is useful for detecting unseen types.

5.2.2 Toward Operational Systems

Beebe and Garfinkel [17] released `Sceadan` just after these studies were conducted. Sceadan is an operational, open source C implementation for file fragment classification using the optimal feature set obtained in their trials. Beebe et al. [16] reports a few possible code efficiency enhancements and a desire to improve classification speed using multi-threaded programming and optimized code. Future improvements could also includes the capability to test and select different feature sets or have the tool test and suggest a feature set based on those with highest information gain. Another addition should be the capability of the tool to adapt over time. This would involve the ability to incorporate new training data as the malware landscape changes and to rank and evaluate input feature vectors since the optimal vector also may change over time.

While Sceadan is an important contribution for file fragment classification, a similar, open source solution is needed for malware detection and other classification problems beyond file type forensics. The development of a tool that allows users to define training and testing data, classes, and experiment with different features and block lengths would greatly enhance research on the use of NLP and information- theoretic features on byte level content and provide operational value to information security analysts monitoring for both seen and unseen malware files.

5.2.3 Final Conclusions

Improved file identification using non-signature based methods is an ever-important area of research in information security. A constantly changing malware landscape, with the proliferation of payload encryption and polymorphic coding, makes malware detection a particularly elusive problem. In this paper, we evaluated the potential of several non-signature based techniques for malware detection, and applied the techniques laterally to another important task in forensics, file type classification. We find that these techniques hold significant potential and value for further research. Overall, information-theoretic and natural language processing features with linear support vector machines and bagged KNN learning algorithms yielded high precision and accuracy rates. Although our results were not as high as Tabish's results [4] for malware detection, they are nonetheless encouraging. For file fragmentation, we were able to achieve significantly higher results in preliminary trials compared to Fitzgerald's [5] recent work. Considering Beebe's [16] recent successes in applying various feature combinations for file fragment identification, we find advanced statistical features useful in machine learning for file classification tasks.

THIS PAGE INTENTIONALLY LEFT BLANK

List of References

[1] S. J. Stolfo, K. Wang, and W.-J. Li, "Towards stealthy malware detection," in *Malware Detection*, ser. Advances in Information Security, M. Christodorescu, S. Jha, D. Maughan, D. Song, and C. Wang, Eds. New York, NY: Springer US, 2007, vol. 27, pp. 231–249.

[2] J. Z. Kolter and M. A. Maloof, "Learning to detect and classify malicious executables in the wild," *Journal of Machine Learning Research*, pp. 2720–2744, 2006.

[3] W.-J. Li, K. Wang, S. J. Stolfo, and B. Herzog, "Fileprints: Identifying file types by n-gram analysis," in *Information Assurance Workshop, 2005. IAW'05. Proceedings from the Sixth Annual IEEE SMC*, New York, NY, 2005, pp. 64–71.

[4] S. M. Tabish, M. Z. Shafiq, and M. Farooq, "Malware detection using statistical analysis of byte-level file content," in *Proceedings of the ACM SIGKDD Workshop on CyberSecurity and Intelligence Informatics*, ser. CSI-KDD '09, New York, NY, 2009, pp. 23–31.

[5] S. Fitzgerald, G. Mathews, C. Morris, and O. Zhulyn, "Using NLP techniques for file fragment classification," *Digital Investigation*, vol. 9, pp. S44–S49, 2012.

[6] M. G. Schultz, E. Eskin, F. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables," in *2001 IEEE Symposium on Security and Privacy*. New York, NY, USA: IEEE, 2001, pp. 38–49.

[7] W. W. Cohen, "Fast effective rule induction," in *Proceedings of the Twelfth International Conference on Machine Learning*. Lake Tahoe, CA, USA: Morgan Kaufmann, 1995, pp. 115–123.

[8] J. O. Kephart and W. C. Arnold, "Automatic extraction of computer virus signatures," in *4th Virus Bulletin International Conference*, Abingdon, UK, 1994, pp. 178–184.

[9] G. J. Tesauro, J. O. Kephart, and G. B. Sorkin, "Neural networks for computer virus recognition," *IEEE Expert*, vol. 11, no. 4, pp. 5–6, 1996.

[10] W.-J. Li, S. Stolfo, A. Stavrou, E. Androulaki, and A. D. Keromytis, "A study of malcode-bearing documents," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, New York, NY, 2007, pp. 231–250.

[11] K. Wang, J. Parekh, and S. Stolfo, "Anagram: A content anomaly detector resistant to mimicry attack," in *Recent Advances in Intrusion Detection*, ser. Lecture Notes

in Computer Science, D. Zamboni and C. Kruegel, Eds. Heidelberg, Germany: Springer Berlin Heidelberg, 2006, vol. 4219, pp. 226–248.

[12] W. C. Calhoun and D. Coles, "Predicting the types of file fragments," *Digital Investigation*, vol. 5, no. Supplement-1, 2008.

[13] C. Veenman, "Statistical disk cluster classification for file carving," in *Third Internation Symposium on Information Assurance and Security, IAS 2007*. Manchester, UK: IEEE, 2007, pp. 393–398.

[14] S. Axelsson, "The normalised compression distance as a file fragment classifier," *Digital Investigation (The Proceedings of the Tenth Annual DFRWS Conference)*, vol. 7, no. 1, pp. S24–S31, Aug 2010.

[15] G. Conti, S. Bratus, A. Shubina, B. Sangster, R. Ragsdale, M. Supan, A. Lichtenberg, and R. Perez-Alemany, "Automated mapping of large binary objects using primitive fragment type classification," *Digital Investigation*, vol. 7, pp. S3–S12, Aug 2010.

[16] N. Beebe, L. Maddox, L. Lishu, and S. Minghe, "Sceadan: Using concatenated n-gram vectors for improved file and data type classification," *IEEE Information Forensics and Security*, vol. 8, no. 9, pp. 1519–1530, Sept 2013.

[17] N. Beebe and S. Garfinkel. (2014). Sceadan github repository. [Online]. Available: https://github.com/nbeebe/sceadan. Accessed May 2014.

[18] R. Farrell and G. Dinolt, "Bringing science to digital forensics with standardized forensic corpora," *Digital Investigation*, vol. 6, pp. s2–s11, 2009.

[19] A. Baranovich. (2013). Vxheavens website. [Online]. Available: http://vx.netlux.org/index.html. Accessed February 2013.

[20] S. Jain, "Malware obfuscator for malicious executables," in *Global Trends in Information Systems and Software Applications*, ser. Communications in Computer and Information Science, P. Krishna, M. Babu, and E. Ariwa, Eds. Heidelberg, Germany: Springer Berlin, 2012, vol. 270, pp. 461–469.

[21] A. Walenstein, R. Mathur, M. R. Chouchane, and A. Lakhotia. The design space of metamorphic malware. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.69.486&rep=rep1&type=pdf

[22] F. Shahzad, M. Shahzad, and M. Farooq, "In-execution dynamic malware analysis and detection by mining information in process control blocks of linux os," *Information Science*, vol. 231, pp. 45–63, May 2013.

[23] E. Filioli. (2013). Operational cryptology and virology lab. [Online]. Available: http://cvo-lab.blogspot.com/2012/04/vx-heavens-archive-spread-knowledge.html. Accessed February 2013.

[24] A. Baranovich. (2013). Vxheavens torrent. [Online]. Available: http://thepiratebay.sx/torrent/7066921/Vx_heavens_collection(all). Accessed February 2013.

[25] Symantec virus definitions. [Online]. Available: http://www.symantec.com/security_response/definitions/download/detail.jsp?gid=sep. Accessed May 2014.

[26] Win32/rbot. [Online]. Available: http://www.microsoft.com/security/portal/threat/encyclopedia/entry.aspx?name=Win32%2fRbot. Accessed May 2014.

[27] Trojan:win32/buzus. [Online]. Available: http://www.f-secure.com/v-descs/trojan_w32_buzus.shtml. Accessed May 2014.

[28] Virus:win32/xorer.x. [Online]. Available: http://www.microsoft.com/security/portal/threat/encyclopedia/entry.aspx?Nameac=Virus%3AWin32%2FXorer.X. Accessed May 2014.

[29] Worm:win32/autorun. [Online]. Available: http://www.microsoft.com/security/portal/threat/encyclopedia/entry.aspx?Name=Worm:Win32/Autorun. Accessed May 2014.

[30] Constructor. [Online]. Available: http://www.f-secure.com/v-descs/constructor_w32_constructor.shtml. Accessed May 2014.

[31] Worm:win32/ngvck.0_40. [Online]. Available: http://www.microsoftsafetyscanner.net/constructorwin32ngvck-0_40-removal/. Accessed May 2014.

[32] Orange: Cross validation. [Online]. Available: http://orange.biolab.si/docs/latest/reference/rst/Orange.evaluation.testing. Accessed May 2014.

[33] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. San Francisco, CA, USA: John Wiley and Sons, Inc., 2005.

[34] N. Smirnov, “Table for estimating the goodness of fit of empirical distributions,” *The Annals of Mathematical Statistics*, vol. 19, pp. 279–81, 1948.

[35] S. Kulback and R. Leibler, “On information and sufficiency,” *The Annals of Mathematical Statistics*, vol. 22, pp. 79–86, 1951.

[36] J. Lin, “Divergence measures based on the shannon entropy,” *IEEE Transaction on Information Theory*, vol. 37, pp. 145–151, 1991.

[37] A. H. Chan and S.-l. Ao, *Advances in Industrial Engineering and Operations Research*. New York, NY, USA: Springer, 2008.

[38] E. Simpson, "Measurement of diversity," *Nature*, vol. 163, p. 688, 1949.

[39] J. Quinlan, "Induction of decision trees," in *Readings in Machine Learning*, J. Shavlik and T. Dietterich, Eds. Hingham, MA, USA: Morgan Kaufmann, 1990, originally published in *Machine Learning* 1:81–106, 1986.

[40] Orange: Classification trees. [Online]. Available: http://orange.biolab.si/docs/latest/reference/rst/Orange.classification.tree. Accessed May 2013.

[41] Orange: My first orange classifier. [Online]. Available: http://orange.biolab.si/doc/ofb/c_nb.htm. Accessed May 2014.

[42] T. Mitchell, *Machine Learning*. New York, NY, USA: McGraw-Hill New York, 1997.

[43] J. Shawe-Taylor and N. Cristianini, *Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge, UK: Cambridge University Press, 2000.

[44] Orange: Support vector machines. [Online]. Available: http://orange.biolab.si/docs/latest/reference/rst/Orange.classification.svm. Accessed May 2014.

[45] Orange ensembles algorithms: Stacking. [Online]. Available: http://orange.biolab.si/docs/latest/reference/rst/Orange.ensemble/#Orange.ensemble.stacking.StackedClassifier. Accessed May 2014.

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California